DESIGN E SVILUPPO DI UN SERVIZIO DI MESSAGGISTICA ISTANTANEA

Relazione di Progetto

Studente:
Bigini Gioele

Numero Matricola:
261990

Corso:
Ingegneria del Software

Supervisore al Progetto:
Prof. Edoardo Bontà

Descrizione

Realizzare un sistema di comunicazione composto di un'applicazione Server ed un'applicazione Client.

Il problema si riduce a due particolari attori:

- Server
- Client

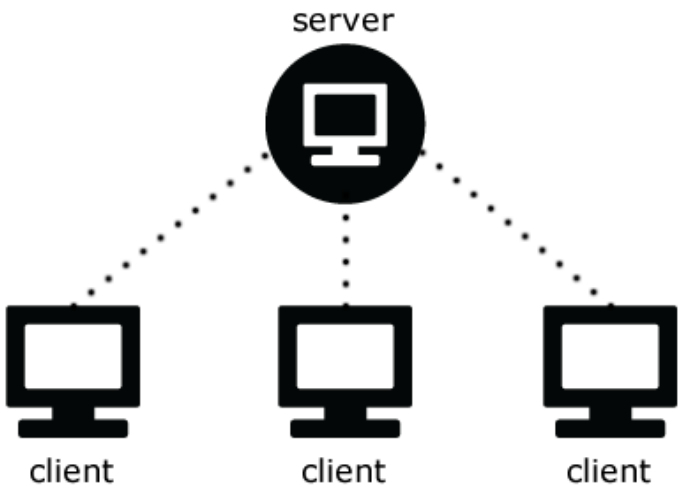
Il Server fornisce servizi di comunicazione ai quali i client sono in grado di accedere per scambiare messaggi. La comunicazione fra i vari client avviene solo ed esclusivamente comunicando con il Server, che prende le sembianze di un intermediario. Inoltre, essa è consentita solamente nel caso in cui i client, a coppie, instaurano una relazione di amicizia, ovvero un sistema con il quale è possibile selezionareda quali client voler ricevere i messaggi.

Le funzionalitànormali offerte ai client sono:

- Servizio per il recupero di informazioni mentre non si era in linea
- Servizio di invio messaggio
- Servizio per l'aggiunta di un nuovo amico
- Servizio per ottenere la propria lista di amici

Il Client consuma i servizi. È l'applicazione con la quale gli utenti interagiscono con il Server. L'interfaccia del Client deve permettere la corretta interazione fra l'utente e quest'ultimo.

Una semplificazione visiva del sistemaè la seguente, dove il Server comunica con i diversi Client connessi ad esso:



Il Server deve, quindi, occuparsi della distribuzione dei servizi correttamente, attraverso le dovute elaborazioni, fra i vari client, possibilmente tenendo aggiornato l'amministratore sulle connessioni in ingresso.

Il Client deve essere in grado di poter dialogare con il Server in maniera corretta, quindi poter comunicare con qualsiasi altro client che faccia parte della propria lista di amici e poter visualizzare la presenza di eventualiinformazioni ricevute mentre non si era in linea.

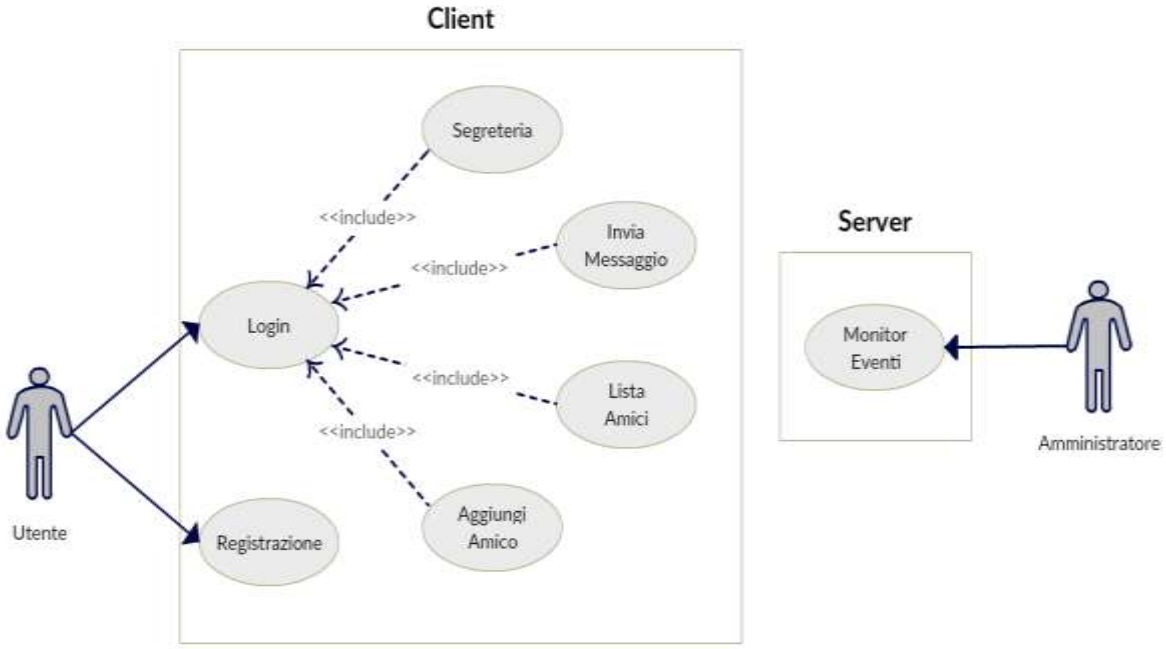
Diagramma dei casi d'uso

Il diagramma dei casi d'uso rappresenta fedelmente le operazioni da implementare in maniera agile e veloce.

Essendo il Server il cuore della comunicazione fra i vari client, è importante disporre, per l'amministratore che avvia il Server,di una sorta di feedback che possa tenerlo al corrente di ciò che succede durante il funzionamento.
Chiameremo questo feedback "Monitor Eventi" e lo rappresenteremo con un'interfaccia a console. Questo monitor di eventi è, quindi, l'interfaccia di output rivolta all'Amministratore ma, si sottolinea, non offrirà funzioni di input particolari perché non richieste al fine progettuale.

Di tutt'altra costruzione è, invece, il Client. L'interfaccia del Client permette all'Utente di usufruire dei servizi, è perciò altamente interattiva. Una volta registrato al servizio, l'Utente potrà effettuare il login e quindi ricevere messaggi lasciati in "segreteria" mentre non c'era, inviarne dei propri, leggere la propria lista delle amicizie e quindi poter aggiungere altri amici.

Si definisce di seguito, il diagramma dei casi d'uso, astraendo da quella che è la comunicazione fra Client e Server.



Relazioni e descrizioni dei casi d'uso

Di seguito vengono analizzati tutti i possibili casi d'uso individuati specificandone Pre-Condizioni e Post-Condizioni.

- L'utente avvia l'applicazione avviando una console che permetterà di scegliere se effettuare l'accesso al servizio attraverso il Login oppure Registrarsi ad esso in caso di mancata registrazione.

  I casi analizzati sono:

  - #1 [Registrazione al Servizio]
  - #2 [Accesso al Servizio]

Caso d'Uso	Registrazione al Servizio
ID	#1
Attore	Utente
Pre-Condizioni	L'utente deve aver avviato l'applicazione Client
Corso d'Azione di Base	1. *L'utente seleziona l'opzione di Registrazione* 2. *L'utente inserisce lo Username con cui verrà identificato nel Servizio* 3. *L'utente inserisce la Password con cui vorrà autenticarsi nel Servizio* 4. *Il programma ripresenta nuovamente l'opzione di Login*
Post-Condizioni	L'utente è registrato correttamente
Percorso Alternativo	L'utente esegue il Login

Caso d'Uso	Accesso al Servizio
ID	#2
Attore	Utente
Pre-Condizioni	L'utente deve aver avviato l'applicazione Client
Corso d'Azione di Base	*1. L'utente seleziona l'opzione di Login* *2. L'utente inserisce il proprio Username* *3. L'utente inserisce la propria Password* *4. Il programma mostra 'interfaccia dei Servizi*
Post-Condizioni	L'utente può visualizzare l'interfaccia dei Servizi
Percorso Alternativo	L'utente si Registra

- Una volta effettuato l'accesso, l'utente è in grado di visualizzare i Servizi Disponibili che vengono elencati di seguito:

  - ❖ Segreteria (messaggi e richieste di amicizia ricevute Offline)
  - ❖ Invia un messaggio a un amico
  - ❖ Ottieni la lista degli amici
  - ❖ Aggiungi un amico

Caso d'Uso	Segreteria
ID	#3
Attore	Utente
Pre-Condizioni	Post-Condizione #2
Corso d'Azione di Base	*1. L'utente seleziona l'opzione Segreteria.* *2. Il programma mostra i messaggi e le richieste di amicizia.*
Post-Condizioni	L'utente può visualizzare messaggi e richieste.
Percorso Alternativo	L'utente non ha messaggi né richieste in segreteria, il programma avvisa l'utente e lo riporta all'interfaccia dei servizi disponibili.

Caso d'Uso	Invia un messaggio a un amico
ID	#4
Attore	Utente
Pre-Condizioni	Post-Condizione #2
Corso d'Azione di Base	*1. L'utente seleziona l'opzione Invia un messaggio a un amico.* *2. L'utente inserisce il nome dell'amico al quale vuole inviare un messaggio.* *3. L'utente inserisce il messaggio che vuole inviare al proprio amico.* *4. Il programma avvisa l'utente se l'invio è andato a buon fine.*
Post-Condizioni	L'utente ha potuto inviare il messaggio al proprio amico.
Percorso Alternativo	L'utente non ha amici e non può inviare alcun messaggio. Il programma avvisa l'utente e lo riporta l'utente all'interfaccia dei servizi disponibili.

Caso d'Uso	Ottieni la lista degli amici
ID	#5
Attore	Utente
Pre-Condizioni	Post-Condizione #2
Corso d'Azione di Base	*1.* L'utente seleziona l'opzione di Ottieni la lista degli amici *2.* Il programma mostra all'utente la propria lista di amici
Post-Condizioni	L'utente può visualizzare la lista degli amici richiesta
Percorso Alternativo	L'utente non ha amici e non può visualizzare la propria lista (ovviamente vuota). Il programma avvisa l'utente e lo riporta all'interfaccia dei servizi disponibili.

Caso d'Uso	Aggiungi un amico
ID	#6
Attore	Utente
Pre-Condizioni	Post-Condizione #2
Corso d'Azione di Base	*1.* L'utente seleziona l'opzione Aggiungi un amico *2.* L'utente inserisce lo username identificativo dell'utente al quale inviare la richiesta *3.* Il programma avvisa l'utente che l'operazione è andata a buon fine
Post-Condizioni	L'utente ha effettuato una richiesta di amicizia
Percorso Alternativo	/

L'analisi e la progettazione è la sezione nella quale si analizzano e giustificano le scelte progettuali finalizzate alla realizzazionedell'applicazione. Pertanto lungo questa parte si analizzeranno i seguenti punti:

- Architettura
- Interfaccia
- Diagramma delle Classi
- Classi
- Design Pattern Utilizzati
- Diagramma delle Sequenze

L'ambiente di sviluppo scelto per il progetto è Microsoft Visual Studio 2013 Community Edition con Framework .NET 4.6.
Il linguaggio di programmazione utilizzato è C#.

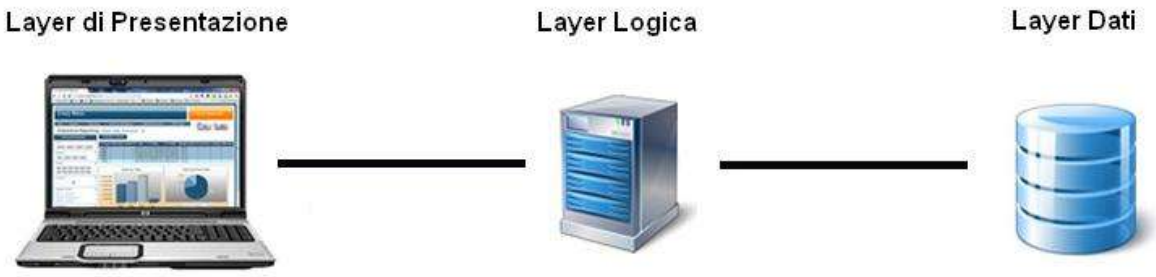
ARCHITETTURA

La soluzione utilizzata per il progetto è una architettura **Multi-Tier**. Questo genere di soluzione permette di separare, organizzandoli in moduli, diversi strati di software adibiti a specifici compiti. In questo modo è possibile disaccoppiare fortemente la logica dall'interfaccia dell'applicazione, oltre a ridurne la complessità in termini di manutenibilità.

La struttura è suddivisa in tre moduli:

- *Interfaccia o Livello di Presentazione*
- *Logica o Livello Applicazione*
- *Dati*

L'interfaccia è relegata alla comunicazione con l'utente, la logica alla gestione ed elaborazione dei contenuti offerti dall'applicazione e infine, ilmodulo dati,alla corretta interazione con le basi di dati.
L'esempio fatto sopra prende il nome di Three-Tier, utilizzatissimo nelle applicazioni Client-Server. Di seguito si presenta una grafica esemplificativa dell'architettura (*l'esempio è specifico della parte Server, nel Clienti moduli si riducono a 2 per l'assenza del layer dedicato ai Dati*).



INTERFACCIA

L'interfaccia del programma è un'interfaccia a Console, presente in entrambe le applicazioni. Essa contiene tutto l'I/O dell'applicazione per cui solo questaclasse dell'interfaccia comunicherà con l'utilizzatore, come richiesto dal progetto.

Di seguito sono analizzate le due interfacce:

- L'interfaccia del *Server* è un'interfaccia che prevede come unica interazione di input un'interazione tale da permetternelo spegnimento.Essa funziona come un monitor, aggiornando l'amministratore delle connessioni in ingresso e segnalando eventuali errori a tempo di esecuzione.

- L'interfaccia del *Client* è più complessa. Essa prevede un menù di Login ad applicazione avviata. L'utente potrà scegliere di effettuare il Login o l'eventuale Registrazione se essa non è ancora stata fatta e, una volta avuto accesso al servizio con il proprio username e relativa password, potrà accedere ai servizi di messaggisticaveri e propri.

DIAGRAMMA DELLE CLASSI

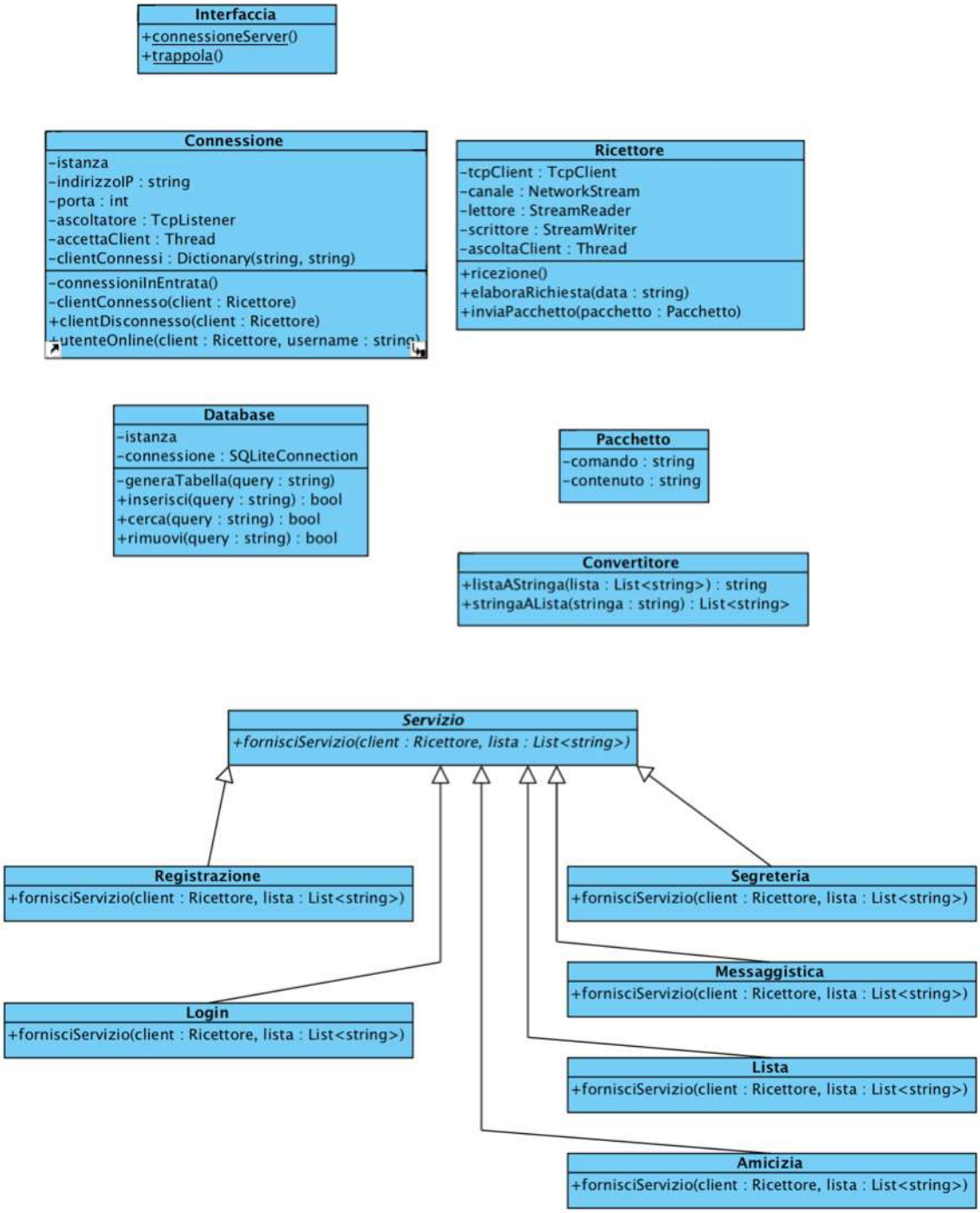
Server

Come anticipato, il Server è divisibile in tre moduli principali che ne costituiscono la "colonna vertebrale" dell'applicazione: l'interfaccia, la logica e il database. La logica è definibile come l'insieme delle classi che costituiranno, con le loro istanze, il comportamento dell'applicazione. Il modulo critico è ovviamente quello riguardante la logica, dove la criticità da affrontareè correlata alla gestione simultanea di connessioni in ingresso e di canali di dialogo Client/Server predisposti per ogni client. Pertanto, imponiamo come vincolo progettuale la separazione fra la logica relativa alla accettazione delle connessioni in ingresso e quella relativa al trattamento dei canali dei singoli client connessi. In altre parole, questa cosa si può tradurre in questo modo:

1. Il server stabilisce la connessione presso un IP ed una Porta ma non apre alcun canale di comunicazione.
2. Per ogni client che richiede l'accettazione della propria connessione (ovvero tenta di collegarsi al server), previa accettazione, predisporre un oggetto (chiamato Ricettore) che si occupi di dialogare solo ed esclusivamente con esso (e quindi non capace di comunicare con altri client).

Sarà, quindi, la controparte del Client sul Server a rispondere alle chiamate remote dei singoli Client Remoti.Successivamente verrà chiarificato, con diagrammi di sequenze, la comunicazione fra le varie istanze.

Nella pagina successiva viene mostrato il diagramma delle Classi.

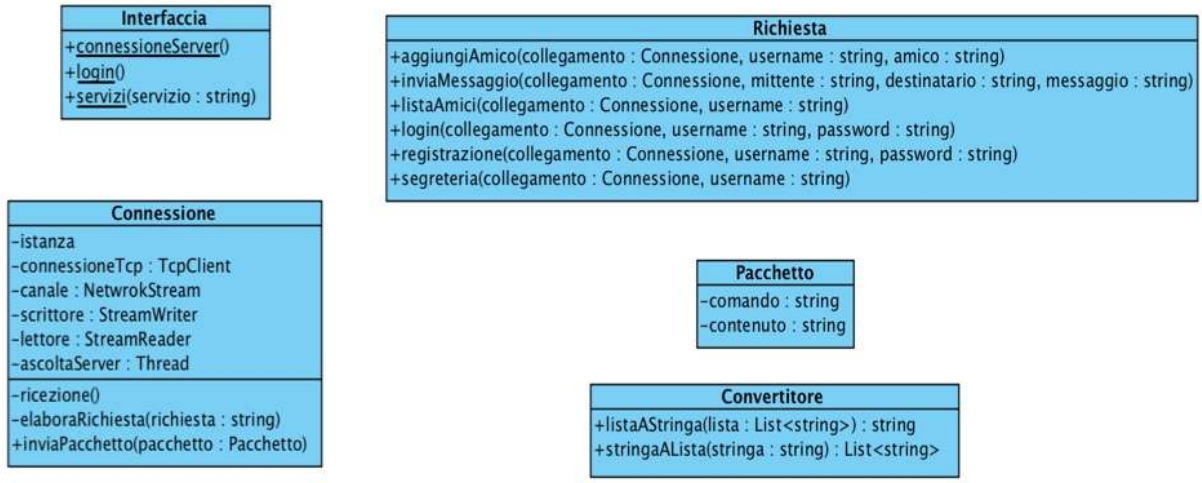

Interfaccia
+connessioneServer()
+trappola()
Connessione
-istanza
-indirizzoIP : string
-porta : int
-ascoltatore : TcpListener
-accettaClient : Thread
-clientConnessi : Dictionary(string, string)
-connessioniInEntrata()
-clientConnesso(client : Ricettore)
+clientDisconnesso(client : Ricettore)
+utenteOnline(client : Ricettore, username : string)
Ricettore
-tcpClient : TcpClient
-canale : NetworkStream
-lettore : StreamReader
-scrittore : StreamWriter
-ascoltaClient : Thread
+ricezione()
+elaboraRichiesta(data : string)
+inviaPacchetto(pacchetto : Pacchetto)
Database
-istanza
-connessione : SQLiteConnection
-generaTabella(query : string)
+inserisci(query : string) : bool
+cerca(query : string) : bool
+rimuovi(query : string) : bool
Pacchetto
-comando : string
-contenuto : string
Convertitore
+listaAStringa(lista : List<string>) : string
+stringaAListA(stringa : string) : List<string>
Servizio
+forniciServizio(client : Ricettore, lista : List<string>)
Registrazione
+forniciServizio(client : Ricettore, lista : List<string>)
Login
+forniciServizio(client : Ricettore, lista : List<string>)
Segreteria
+forniciServizio(client : Ricettore, lista : List<string>)
Messaggistica
+forniciServizio(client : Ricettore, lista : List<string>)
Lista
+forniciServizio(client : Ricettore, lista : List<string>)
Amicizia
+forniciServizio(client : Ricettore, lista : List<string>)

Client

Nel Client, a differenza del Server, si ha una complessità notevolmente ridotta. I moduli dell'applicazione, in assenza del Database, si riducono a Interfaccia e Logica. Nel caso del Client, il modulo dell'interfaccia è altamente interattivo, pertanto si avrà un grande coinvolgimento della stessa nel funzionamento del programma. In particolare si riassume:

- L'interazione con l'utente è totalmente contenuta nella classe Interfaccia.
- Lo stabilimento della connessione al Server è compito della classe Connessione, come anche le operazioni di ricezione nel canale dati.

In questo modo si garantisce una forte separazione fra i moduli Interfaccia e Logica.



CONSIDERAZIONI PROGETTUALI

Generalmente durante la creazione di un ambiente Client-Server si fa in modo di aderire a determinati standard. Nel caso attuale, sarebbe opportuno aderire allo standard XML, standard largamente utilizzato nei servizi web. Per non aumentare ulteriormente la complessità del progetto (che lentamente si va delineando in questa sezione) si evita, però, di inserire un altro obbiettivo da raggiungere, legato questa volta alla conformità. Visto che l'applicazione, al fine progettuale, non dovrà comunicare con servizi diversi (GSOAP e simili), ma solo con altri client sviluppati nel medesimo linguaggio, si evita di rendere conforme l'applicazione al suddetto standard.

Per tanto potremo semplicemente considerare che un pacchetto inviato nel canale di comunicazione sia semplicemente composto da due attributi:

- Comando: attributo che specifica il tipo di richiesta.

- Contenuto: attributo che ne specifica i dati necessari per fare in modo che il server esaudisca con successo la richiesta. Se il comando fosse, ad esempio, "Aggiungi un amico", il server deve sapere chi sono e chi voglio aggiungere, ovvero il mittente ed il destinatario.

CLASSI

Di seguito vengono descritte le classi che compongono le due applicazioni del progetto.

Server

Layer di Presentazione

- Interfaccia:l'interfaccia è la classe con la quale l'amministratore che avvia il server può interfacciarsi.
  L'interfaccia permette di rimanere aggiornato sugli eventi che occorrono a Server avviato e scappatoie per la corretta chiusura dell'intero sistema.

  - connessioneServer(): il metodo consente lo stabilimento della connessione del Server istanziando un oggetto della classe Connessione.
  - trappola(): il metodo è adibito per la cattura del comando (o combinazione) per la chiusura dell'applicazione. Verosimilmente si catturerà la combinazione CTRL-C.

Layer di Logica

- Connessione: la classe si occupa di stabilire la connessione presso un indirizzo IP specifico e relativa porta. È quindi la classe che si occupa di restare in ascolto per le connessioni in ingresso.

  - connessioneEntrata(): il metodo si occupa di restare in ascolto per l'accettazione delle connessioni in ingresso. Un flusso di esecuzione dovrà mantenere attivo il metodo continuamente, pertanto potrebbe essere necessario un thread.

- clientConnesso():il metodo ha il compito di aggiornare il dizionario della classe con il nuovo client connesso nel momento in cui una connessione viene accettata.
- clientDisconnesso(): il metodo ha il compito di aggiornare il dizionario della classe rimuovendo il client disconnesso nel momento in cui la connessione con questo venga meno.
- utenteOnline(): quando un utente accede al servizio (fare attenzione alla distinzione fra client connesso e utente connesso), il metodo entra in azione per aggiornare il dizionario, associando i dati di connettività del client con lo username.

- Ricettore: nel server la gestione della rete dei singoli client non avviene nella classe connessione per meglio separare i due concetti, quello di gestione delle connessioni in ingresso e quello della gestione della singola connessione con un determinato client. Le istanze di Ricettore sono, in pratica, i mezzi di comunicazione con i client remoti, uno per ogni client. La comunicazione, in questo modo, è concorrente e garantita, lasciando come unico collo di bottiglia l'accesso al database.

  - ricezione(): metodo per controllare la presenza di dati nella rete.
  - elaboraRichiesta():metodo per interpretare i dati ricevuti in rete.
  - inviaPacchetto(): metodo per l'invio dei pacchetti in rete.

- Pacchetto: questa classe definisce l'oggetto pacchetto. Quando stabiliamo una connessione TCP i pacchetti vengono trasferiti da un End-Point all'altro, ma per poterli spedire è ovviamente necessario definirli.

- Convertitore:Il contenuto dei pacchetti sarà un determinato tipo di dato, ad esempio una stringa. Il nome della classe è un chiaro riferimento al compito che deve svolgere: convertire il tipo di dato in qualcosa di più facile manipolazione, ad esempio una lista. Viceversa, si potrebbe voler agilmente costruire da una lista di stringhe, una stringa che sia conforme

a quelle accettate nei pacchetti. Per questo motivo la classe implementa i seguenti metodi:

- listaAStringa(): il metodo, presa una lista, costruisce una stringa di formato coerente con il formato del pacchetto.
- stringaALista(): il metodo preso il contenuto del pacchetto, ad esempio una stringa di lunghezza indefinita, ne costruisce una lista di stringhe di più facile manipolazione.

- Servizio:la classe è astratta e contiene un metodo astratto utilizzato per fornire un determinato servizio. Le seguenti classi sono le classi derivate da essa:
  - Amicizia
  - Lista
  - Login
  - Messaggistica
  - Registrazione
  - Segreteria

- Amicizia:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa all'aggiunta di un amico da parte di un richiedente. Questa classe ha la capacità di comunicare con la base di dati per mantenere in memoria le relazioni fra gli utenti.

- Lista:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa alla visualizzazione della propria lista di amici, ovvero la lista degli utenti che hanno consentito il dialogo in seguito alla richiesta di amicizia. Questa classe ha la capacità di comunicare con la base di dati per ricavare i dati delle amicizie relative all'utente richiedente.

- Login:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa al Login di utente al servizio. Questa classe ha la capacità di comunicare con la base di dati per ricavare i dati da confrontare con quelli inseriti dal client che tenta l'operazione di login.

- Messaggistica:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa allo scambio di messaggi fra i client.Questa classe ha la capacità di comunicare con la base di dati per mantenere in memoria eventuali messaggi che non possono essere immediatamente inoltrati a utenti non in linea.

- Registrazione:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa alla registrazione di un nuovo utente al servizio. Questa classe ha la capacità di comunicare con la base di dati per gestire le utenze.

- Segreteria:classe derivata di Servizio. La classe permette la risoluzione della problematica relativa alla gestione delle richieste non inoltrate. Consentirà, pertanto, di ricevere richieste e messaggi in un momento differente dal tempo reale di invio.

Layer Dati

- Database: questa classe è il modulo che gestisce l'accesso al database ed è in grado di operare sulla stessa, fornendo i dati allo strato superiore.

  - generaTabella(): metodo per generare le tabelle della base di dati.
  - inserisci(): metodo per inserire dati nella tabella della base di dati.
  - cerca(): metodo per effettuare una ricerca all'interno delle tabelle del database.
  - rimuovi(): metodo per effettuare la rimozione di un dato all'interno della base di dati.

Client

Layer di Presentazione

- Interfaccia:l'interfaccia è la classe con la quale l'utente che avvia l'applicazione Client può interfacciarsi. Essa permette di interagire remotamente con il Server permettendo l'utilizzo dei vari servizi. Essa implementa anche tutte le azioni I/O ed è, quindi, in netta separazione con il Layer di Logica attraverso il quale è in grado di richiedere tutti i compiti ad esso non assegnati.

  - connessioneServer(): il metodo si occupa di effettuare la connessione con il server remoto.
  - login(): il metodo costituisce la prima fase di interazione con l'utente. Richiamando il metodo vengono presentate le funzioni di di accesso al servizio.
  - servizi(): il metodo viene richiamato nel momento in cui un utente esegue l'accesso al servizio. Richiamando il metodo si potrà accedere ai servizi a disposizione.

Layer di Logica

- Connessione:la classe differisce da quella nel server. I compiti da svolgere sono differenti diversi al server per cui è logico che sia così. I metodi principali si occupano di mantenere aggiornato il client riguardo l'attività della rete.

  - ricezione(): metodo con il quale si controlla la presenza di dati nella rete.
  - elaboraRichiesta():metodo attraverso il quale si interpretano i pacchetti ricevuti dal server.
  - inviaPacchetto(): metodo con il quale vengono spediti i pacchetti.

- Pacchetto: questa classe definisce l'oggetto pacchetto. Quando stabiliamo una connessione TCP i pacchetti vengono trasferiti da un end-point all'altro, ma per poterli spedire è ovviamente necessario definirli.

- Convertitore:Il contenuto dei pacchetti sarà un determinato tipo di dato, ad esempio una stringa. Il nome della classe è un chiaro riferimento al compito che deve svolgere: convertire il tipo di dato in qualcosa di più facile manipolazione, ad esempio una lista. Viceversa, si potrebbe voler agilmente costruire da una lista di stringhe, una stringa che sia conforme a quelle accettate nei pacchetti. Per questo motivo la classe implementa i seguenti metodi:

  - listaAStringa(): il metodo, presa una lista, costruisce una stringa di formato coerente con il formato del pacchetto.
  - stringaAListA(): il metodo preso il contenuto del pacchetto, ad esempio una stringa di lunghezza indefinita, ne costruisce una lista di stringhe di più facile manipolazione.

- Richiesta:La classe richieste permette agilmente di sfruttare i metodi di invio delle richieste al server.

  - aggiungiAmico(): metodo utilizzato per richiedere di aggiungere un amico al server.
  - inviaMessaggio(): metodo utilizzato per richiedere di inviare un messaggio al server.
  - listaAmici(): metodo utilizzato per richiedere la propria lista di amici al server.
  - login(): metodo utilizzato per richiedere eseguire il login al servizio, ovvero eseguire la propria autenticazione.
  - registrazione(): metodo utilizzato registrarsi al servizio e di conseguenza, per poter, in seguito, eseguire il login.

- segreteria(): metodo utilizzato per richiedere al server eventuali messaggi o richieste di amicizia rimaste in sospeso perché offline al momento del ricevimento.

DESIGN PATTERN UTILIZZATI

Il progetto utilizzerà i Design Pattern utili alla risoluzione di problemi tipici della OOP. In particolare il Singleton, che permette assoluto controllo sulle istanze di una classe. Il Pattern si applicherà alla classe Connessione (Client e Server) e alla classe Database (solo Server).
Di seguito viene descritto il funzionamento del Design pattern Singleton.

SINGLETON

Il Singleton è un fondamentale Design Pattern creazionale il cui obbiettivo è creare una e una sola istanza di una classe fornendo un punto di accesso globale ad essa.

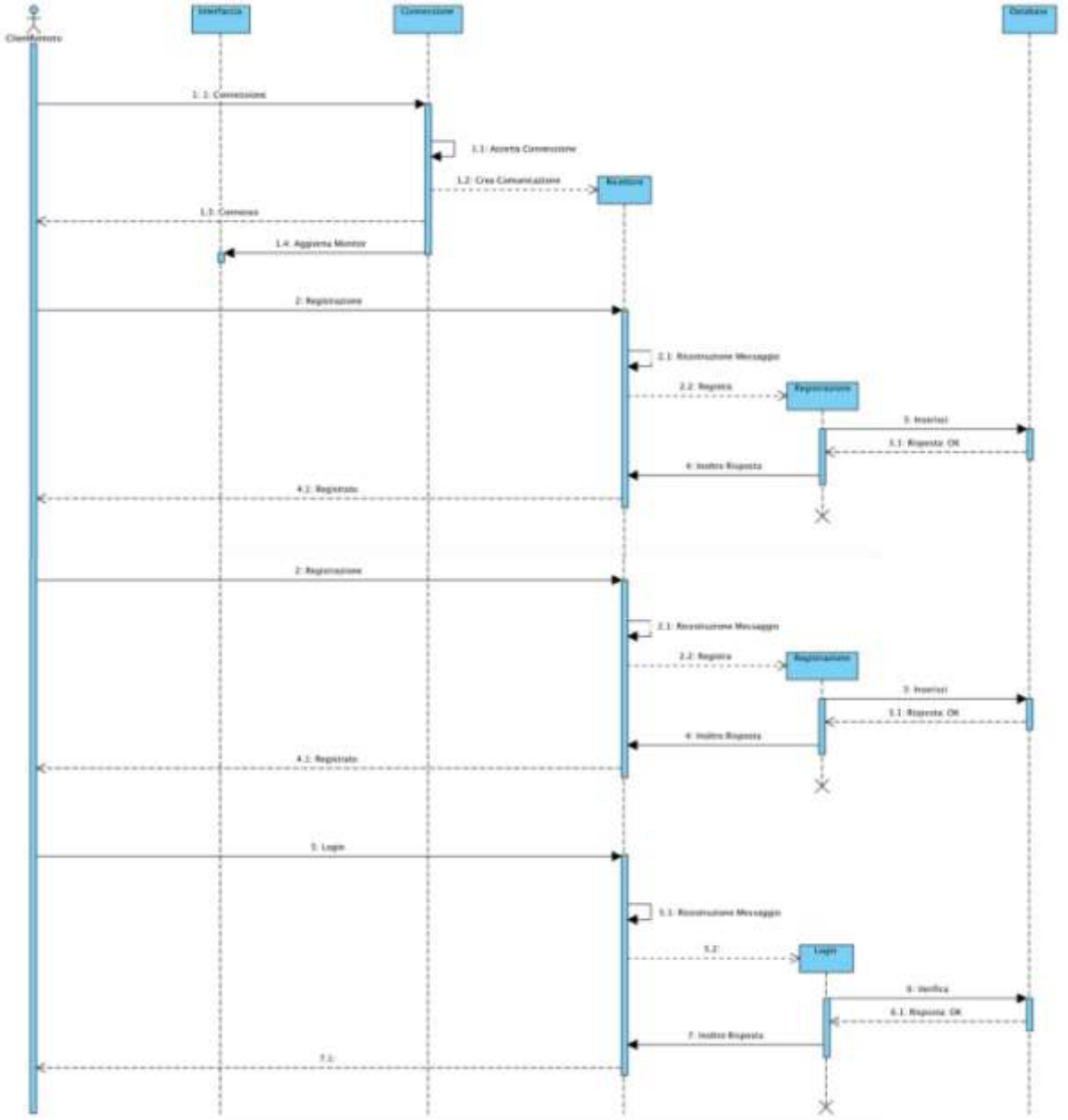
Le proprietà di tale Pattern sono:

- Il costruttore deve essere unico e privato per evitare che vengano create istanze della classe in maniera diretta
- La classe deve contenere un metodo (solitamente chiamato getter) che permetta di creare un'istanza della classe una e una sola volta.

Tale pattern utilizza, poi, un secondo pattern chiamato "Lazy Initialization" che prevede di creare l'istanza della classe al primo utilizzo.

DIAGRAMMA DELLE SEQUENZE

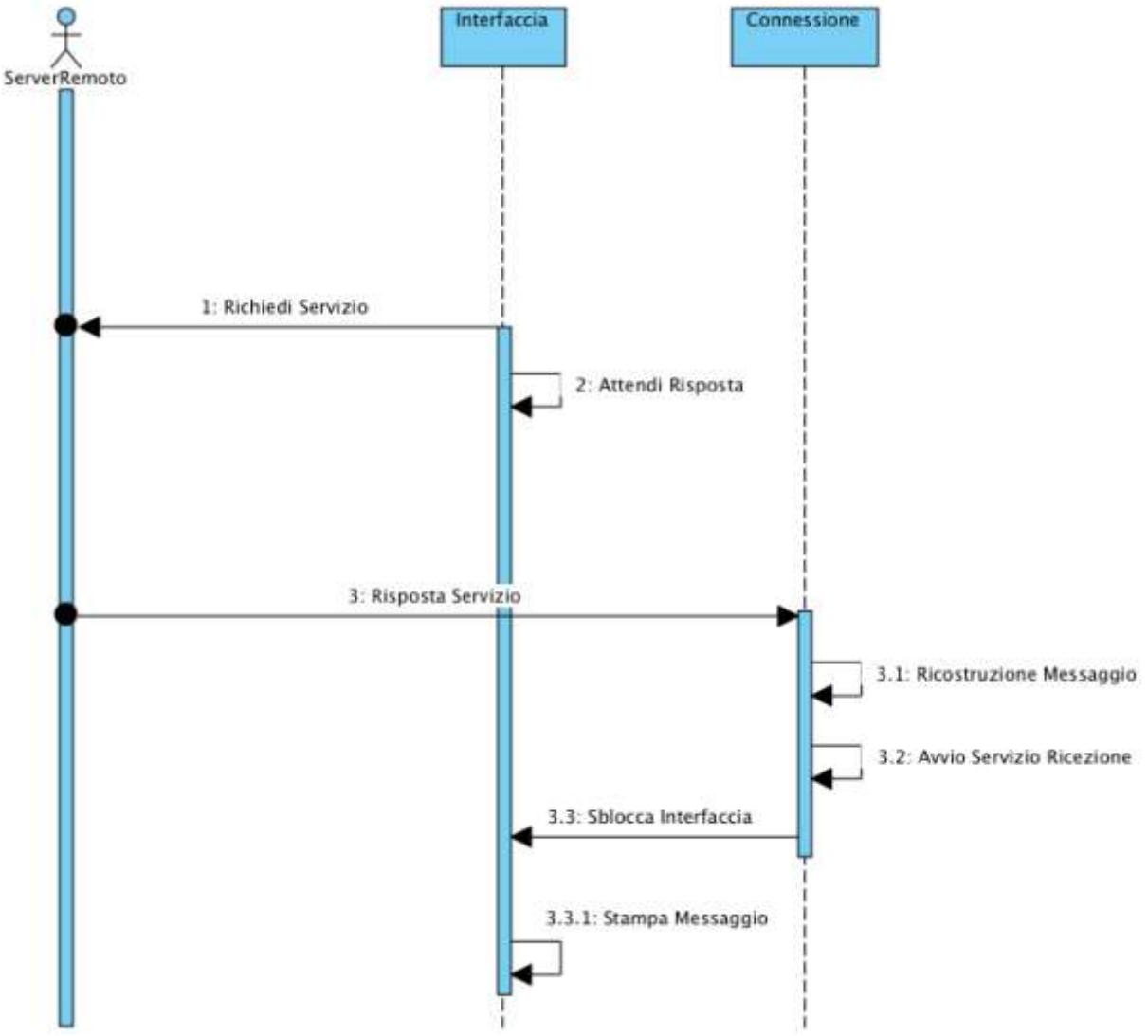
Server



Client

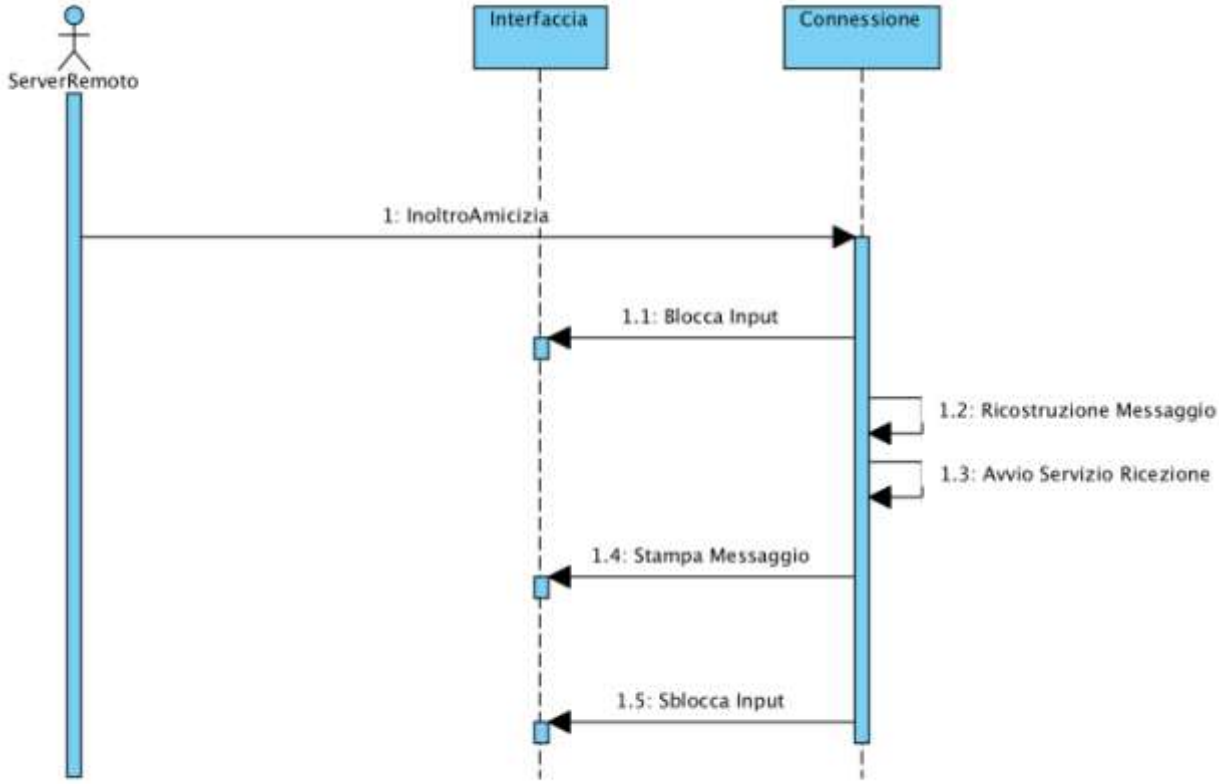
Nei seguenti diagrammi si descrivono due principali comportamenti. Il comportamento del Client che invia un messaggio e attende la risposta da parte del Server (comunicazione sincrona) e la ricezione di messaggi "casuali", dove il sinonimo di "casuali" è "inattesi", pertanto una ricezione asincrona.

COMUNICAZIONE SINCRONA:



NOTA: *I punti neri indicano il fatto che il Client non saprà se il messaggio raggiungerà il Server, né se il Server in caso di ricezione, risponderà. La scelta di utilizzare come attore un'entità rappresentata da un'applicazione è utile al fine della spiegazione del funzionamento.*

COMUNICAZIONE ASINCRONA:



NOTA:*InoltroAmicizia è un esempio di servizio che avvisa l'utente che un determinato utente (non sé stesso) l'ha appena aggiunto agli amici. L'utente "destinatario della amicizia" potrebbe essere aggiunto in qualsiasi momento per cui è impossibile poter prevedere quando il messaggio verrà inviato dal server. Il diagramma mette in luce le criticità da affrontare. In fase di ricezione, infatti, l'utente potrebbe utilizzare servizi e essere interrotto da questa chiamata forzatamente. Sarà importante, pertanto, bloccare l'interfaccia di input all'utente preventivamente e, ancora meglio, trovare un espediente per fare in modo che questo genere di chiamate facciano la loro apparizione solo quando l'utente non sta utilizzando alcun servizio.*

SERVER

Interfaccia.cs

```
using System;
using System.Threading;

namespace Server
{
  /* Tutta la comunicazione con l'amministratore è contenuta in questa classe */
  static class Interfaccia
  {
    /* Funzione Main del programma */
    static int Main(string[] args)
    {
      /* Creazione evento di cattura CTRL-C */
      Console.CancelKeyPress += new ConsoleCancelEventHandler(trappola);

      /* Messaggio di Benvenuto all'Amministratore */
      testoApertura();

      /* Stabilimento connessione Server */
      connessioneServer();

      return 0;
    }

    /* METODI */
    /* Metodo contenente il messaggio di Benvenuto */
    static void testoApertura()
    {
      Console.WriteLine("\t\t  * * * * * * * * * * * * * * * * * * * * * *");
      Console.WriteLine("\t\t  *   Progetto di Ingegneria del Software   *");
      Console.WriteLine("\t\t  *    Software Messaggistica Istantanea    *");
      Console.WriteLine("\t\t  *    Bigini Gioele - Matricola: 261990    *");
      Console.WriteLine("\t\t  *          Versione Software: 1.0          *");
      Console.WriteLine("\t\t  * * * * * * * * * * * * * * * * * * * * * *\n\n");
    }

    /* Metodo relegato allo stabilimento della connessione del Server presso un IP e una Porta*/
    static void connessioneServer()
    {
      Console.WriteLine(" -> Stabilisco la connessione al Database\n");
      /* Creazione dell'istanza di Database */
      Database database = Database.Istanza;
      Console.WriteLine(" <> Connessione al Database stabilita\n");

      Console.WriteLine(" -> Stabilisco la connessione del Server\n");
      /* Connessione è una classe Singleton con Lazy Initialization     *
          * Pertanto la prima chiamata istanza l'oggetto (una volta sola ) */
      Console.WriteLine(" <> Connessione stabilita presso {0}:{1}\n\n",
                                      Connessione.Istanza.indirizzoIP,
                                      Connessione.Istanza.porta);

      Console.WriteLine("\t\t *  Benvenuto Amministratore - Server Online  *\n\n");
      Console.WriteLine(" -> Ricorda! La procedura di spegnimento si attiva con CTRL-C\n");
      Console.WriteLine(" -> Resto in attesa di Connessioni...\n\n");
    }

    /* Metodo eseguito in seguito alla cattura della combinazione CTRL-C */
    static void trappola(object sender, ConsoleCancelEventArgs pressione)
    {
      /* Messaggio a schermo di conferma spegimento */
      Console.Clear();
      testoApertura();
      Console.WriteLine(" <!> Rilevata richiesta di spegnimento del Server <!>\n");
      Console.WriteLine(" -> Vuoi davvero spegnere il Server? [S/N]\n");

      ConsoleKeyInfo cki = new ConsoleKeyInfo();
```

```
      /* Cattura della scelta dell'utente                                          *
       * Questo procedimento consente di prelevare gli input senza che l'utente prema invio *
       * Se l'utente preme S, comincia la procedura di spegnimento                 *
       * Se l'utente preme N, la procedura viene interrotta,il server continua a funzionare */
      while (cki.Key != ConsoleKey.S && cki.Key != ConsoleKey.N)
      {
        cki = Console.ReadKey(true);

        switch (cki.Key)
        {
          /* S */
          case ConsoleKey.S:
              Console.WriteLine(" <!> Attendere prego. Procedura di spegnimento avviata <!>\n");
              Console.WriteLine(" -> Fatto. Arrivederci Amministratore =D");
              Thread.Sleep(1000);
              Environment.Exit(0);
              break;
          /* N */
          case ConsoleKey.N:
              Console.Clear();
              testoApertura();
              Console.WriteLine(" -> Stabilisco la connessione del Server\n");
              Console.WriteLine(" <> Connessione stabilita presso {0}:{1}\n\n",
                                          Connessione.Istanza.indirizzoIP,
                                          Connessione.Istanza.porta);
              Console.WriteLine("\t\t *  Benvenuto Amministratore - Server Online  *\n\n");
              Console.WriteLine(" -> Ricorda! La procedura di spegnimento si attiva con CTRL-C\n");
              Console.WriteLine(" -> Resto in attesa di Connessioni...\n\n");
              pressione.Cancel = true;
              break;
        }
      }
    }

    /* Metodo per la stampa di un eccezione in qualsiasi zona del codice */
    public static void stampaMessaggio(string messaggio)
    {
      Console.WriteLine(messaggio);
    }
  }
}
```

Connessione.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Server
{
  class Connessione
  {
    /* VARIABILI */
    private static Connessione istanza; // Istanza Singleton
    public string indirizzoIP { get; private set; } // Proprietà per l'indirizzo IP
    public int porta { get; private set; } // Proprietà per la porta utilizzata
    private TcpListener ascoltatore; // Listener in ascolto delle connessioni
    private Thread accettaClient; // Thread che si occupa di accettare i client

    /* Dizionario contenente i dati relativi alle connessioni stabilite     *
     * Campo chiave : Oggetto Cliente                                       *
     * Campo valore : Stringa Username (solo in caso di accesso al Servizio */
    public Dictionary<Ricettore, string> ClientConnessi { get; private set; }
```

```
    /* COSTRUTTORE */
    private Connessione()
    {
```

Queste due righe di codice determinano l'IP e la porta in cui si stabilisce la connessione

```
      /* Assegnazione Indirizzo IP e Porta del Server */
      indirizzoIP = "127.0.0.1";
      porta = 5302;

      /* Assegnazione di Dizionario */
      ClientConnessi = new Dictionary<Ricettore,string>();

      /* Assegnazione e Avvio del Listener */
      try
      {
        ascoltatore = new TcpListener(IPAddress.Parse(indirizzoIP), porta);
        ascoltatore.Start();

        /* Assegnazione e Avvio del thread relegato ad accettare i Client */
        accettaClient = new Thread(connessioniInEntrata);
        accettaClient.Start();
      }
      catch (SocketException)
      {
        Interfaccia.stampaMessaggio(" <!> Non è possibile stabilire la connessione <!>\n");
        Interfaccia.stampaMessaggio(" <!> Controlla che un'istanza del Server non sia gia' Online
<!>\n");
        Interfaccia.stampaMessaggio(" -> Chiusura automatica in 10 secondi");
        Thread.Sleep(10000);
        Environment.Exit(0);
      }
    }

    /* METODI */
    /* Metodo per creazione dell'istanza Singleton*/
    public static Connessione Istanza
    {
      get
      {
        if (istanza == null)
        {
          istanza = new Connessione();
        }
        return istanza;
      }
    }
```

Questo metodo é sempre attivo. Un flusso di esecuzione si occupa di tenerlo vivo. Quello che fa è accettare le connessioni in ingresso e associarci un oggetto di tipo Ricettore che servirà per la comunicazione con il Client specifico.

```
    /* Metodo per il controllo delle connessioni in ingresso */
    private void connessioniInEntrata()
    {
      do
      {
        try
        {
          /* Con una chiamata Bloccante attendo la connessione di nuovi Client *
           * Creo un oggetto che fungerà da STUB                              */
          Ricettore client = new Ricettore(ascoltatore.AcceptTcpClient());
          clientConnesso(client); // Aggiungo il client nel dizionario
        }
        catch (Exception)
        {
          Interfaccia.stampaMessaggio(" (!) Si e' verificato un errore con l'accettazione di un
client\n");
        }
      } while (true);
    }
```

```
    /* Metodo per l'inserimento del Client connesso nel Dizionario */
    private void clientConnesso(Ricettore client)
    {
      /* Se il client non è già stato inserito nel Dizionario, Inseriscilo    *
        * Se il client risulta già nel dizionario, segnala un malfunzionamento */
      if (!ClientConnessi.ContainsKey(client))
      {
        ClientConnessi.Add(client, null);
        Interfaccia.stampaMessaggio(" (Connesso) Client Connesso: " +
client.TcpClient.Client.RemoteEndPoint + "\n");
      }
      else
        Interfaccia.stampaMessaggio("Errore in fase di inserimento nel dizionario");
    }

    /* Metodo per la rimozione di un Client dal dizionario              *
      * public perchè l'avviso di rimozione giungerà dall'oggetto Client */
    public void clientDisconnesso(Ricettore client)
    {
      /* Lock per maneggiare correttamente il Dizionario */
      lock (ClientConnessi)
        /* Rimozione del Client se esisistente, altrimenti errore */
        if (ClientConnessi.ContainsKey(client))
        {
          ClientConnessi.Remove(client);
          Interfaccia.stampaMessaggio(" (Disconnesso) Client Disconnesso: " +
client.TcpClient.Client.RemoteEndPoint + "\n");
        }
        else
          Interfaccia.stampaMessaggio("Il Client" + client.TcpClient.Client.RemoteEndPoint + "ha
richiesto la rimozione, ma non esiste!");
    }

    /* Metodo per l'aggiornamento del dizionario se il Client accede al Servizio*/
    public bool utenteOnline(Ricettore client, string username)
    {
      bool esito;

      /* Lock per maneggiare correttamente il Dizionario */
      lock (ClientConnessi)
        /* Aggiornamento del Client se esistente */
        if (ClientConnessi.ContainsKey(client))
          if (!ClientConnessi.ContainsValue(username))
          {
            ClientConnessi[client] = username;
            esito = true;
          }
          else
            esito = false;
        else
          esito = false;

      return esito;
    }

    /* Metodo per la ricerca nel Dizionario per valore (non per chiave) */
    public Ricettore cercaInDizionario(string username)
    {
      Ricettore clientRitorno = null;

      /* Lock per maneggiare correttamente il Dizionario */
      lock (ClientConnessi)
        /* Ricerco per valore */
        if (ClientConnessi.ContainsValue(username))
          /* Estraggo la chiave */
          clientRitorno = ClientConnessi.FirstOrDefault(x => x.Value == username).Key;

      return clientRitorno;
    }
  }
}
```

Ricettore.cs

```
using Server.Servizi;
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace Server
{
  class Ricettore
  {
    /* VARIABILI */
    public TcpClient TcpClient { get; private set; } // Posizione del Client
    private Thread ascoltaClient; // Thread per attività
    private NetworkStream canale; // Canale trasmissione dati
    private StreamReader lettore; // Lettore del canale
    private StreamWriter scrittore; // Scrittore del canale

    /* COSTRUTTORE */
    public Ricettore(TcpClient client)
    {
      /* Memorizzazione della posizione del Client */
      TcpClient = client;

      /* Apertura Canale e assegnazione strumenti di Lettura e Scrittura nello stesso */
      canale = client.GetStream();
      lettore = new StreamReader(canale);
      scrittore = new StreamWriter(canale);

      /* Creazione e Avvio di un Thread dedicato alla comunicazione Client Remoto associato */
      ascoltaClient = new Thread(ricezione);
      ascoltaClient.Start();
    }

    /* METODI */
```

Questo metodo é sempre attivo. Un flusso di esecuzione si occupa di tenerlo vivo. Quello che fa è accertarsi continuamente che il canale di comunicazione con il client sia mantenuto sotto controllo i messaggi in ricezione.

```
    /* Metodo per la ricezione dei dati nel canale dal Client Remoto*/
    private void ricezione()
    {
      bool condizione = true; // Condizione per il ciclo
      List<int> listaCaratteri = new List<int>(); // Lista dei caratteri acquisiti in ricezione
      int valore = new int(); // Variabile per valore acquisito nel canale

      /* Esegui le istruzioni se il Client è connesso             *
       * Interrompi il ciclo in caso di Disconnessione del Client */
      do
      {
        try
        {
          /* Acquisizione del valore */
          valore = lettore.Read();

          /* Se non è il valore terminatore */
          if (valore != 0)
              listaCaratteri.Add(valore); // Aggiunta del valore alla lista
          /* Altrimenti, è il valore terminatore, composizione del messaggio ricevuto */
          else
          {
            var messaggio = new StringBuilder();

            /* Costruzione del messaggio ricevuto - Conversione degli interi in caratteri */
            for (int i = 0 ; i < listaCaratteri.Count ; i++)
              messaggio.Append(Convert.ToChar(listaCaratteri[i]));
```

```
            /* Invio del messaggio ricevuto per l'elaborazione */
            elaboraRichiesta(messaggio.ToString());

            /* Svuoto la lista degli interi */
            listaCaratteri.Clear();
          }
        }
        catch (IOException) // Se il Client si disconnette
        {
          /* Rimozione Client dal Dizionario */
          Connessione.Istanza.clientDisconnesso(this);
          /* Uscita dal Ciclo */
          condizione = false;
        }
      } while (condizione);
    }

    /* Metodo per l'elaborazione della richiesta ricevuta dal Client Remoto */
    private void elaboraRichiesta(string data)
    {
      try
      {
        /* Lettura del comando */
        var pacchetto = new Pacchetto(data);
        string comando = pacchetto.Comando;

        /* Conversione del messaggio in lista */
        Convertitore converti = new Convertitore();
        var lista = converti.stringaALista(pacchetto.Contenuto);

        /* Smistamento della richiesta al servizio */
        switch (comando)
        {
          case "Login":
            new Login().forniciServizio(this, lista);
            break;
          case "Registrazione":
            new Registrazione().forniciServizio(this, lista);
            break;
          case "AggiungiAmico":
            new Amicizia().forniciServizio(this, lista);
            break;
          case "Messaggio":
            new Messaggistica().forniciServizio(this, lista);
            break;
          case "ListaAmici":
            new Lista().forniciServizio(this, lista);
            break;
          case "Segreteria":
            new Segreteria().forniciServizio(this, lista);
            break;
          default:
            Interfaccia.stampaMessaggio("Pacchetto Invalido");
            break;
        }
      }
      catch (Exception)
      {
        Interfaccia.stampaMessaggio(" (!) Il pacchetto ricevuto è Invalido\n");
      }
    }

    /* Questo Metodo, con il successivo, permette l'invio dei pacchetti nel
    NetworkStream */
    public void InviaPacchetto(Pacchetto pacchetto)
    {
      /* Invio nel NetworkStream */
      scrittore.Write("{0}:{1}\0", pacchetto.Comando, pacchetto.Contenuto);
      /* Svuoto lo Stream */
      scrittore.Flush();
    }
  }
}
```

Convertitore.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
  class Convertitore
  {
    /* METODI */
    /* Metodo per la conversione da lista a stringa */
    public string listaAStringa(List<string> lista)
    {
      string stringaOttenuta; // stringa di ritorno

      /* Controlla che la lista non sia vuota */
      if (lista.Count != 0)
      {
        /* utilizzo un boolean per creare un comportamento diverso sul *
          * primo elemento della lista */
        bool primoElemento = true;

        /* Creo una variabile di tipo StringBuilder */
        var stringa = new StringBuilder();

        /* Costruisco la stringa di questo tipo: *
          * Stringa1,Stringa2,Stringa3...         */
        foreach (var elemento in lista)
        {
          if (primoElemento)
          {
            stringa.Append(elemento);
            primoElemento = false;
          }
          else
          {
            stringa.Append(string.Format(",{0}", elemento));
          }
        }

        /* StringBuilder non è tipo string ma è facilmente rapportabile */
        stringaOttenuta = stringa.ToString();
      }
      else
      {
        stringaOttenuta = null; // Se vuota il valore è null
      }

      /* Ritorno della stringa */
      return stringaOttenuta;
    }

    /* Metodo per la conversione da Stringa a Lista */
    public List<string> stringaALista(string stringa)
    {
      /* Variabile che ospiterà la lista */
      List<string> lista = new List<string>();

      /* Se la stringa non è null o vuota */
      if (!string.IsNullOrEmpty(stringa))
      {
        /* Prova a comporre la stringa */
        try
        {
          foreach (string elemento in stringa.Split(','))
          {
            lista.Add(elemento);
          }
        }
```

```
        catch (Exception)
        {
          lista = null; // in caso di errore invia lista vuota
        }
      }
      else
      {
        lista = null; // La stringa è null, allora la lista è null
      }

      /* Ritorno della lista */
      return lista;
    }
  }
}
```

Pacchetto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
  /* Classe per la generazione dei pacchetti */
  class Pacchetto
  {
    /* VARIABILI */
    /* I pacchetti contengono i parametri                *
      * Comando   : Nome del servizio richiesto o offerto *
      * Contenuto : Contenuto devoluto                     */
    public string Comando { get; private set; }
    public string Contenuto { get; private set; }

    /* COSTRUTTORI */
    /* 2 Costruttori da utilizzare secondo l'uso */
    /* Costruzione di un pacchetto direttamente con i parametri assegnati */
    public Pacchetto(string comando, string contenuto)
    {
      Comando = comando;
      Contenuto = contenuto;
    }

    /* Costruttore di un pacchetto attraverso una stringa */
    public Pacchetto(string dati)
    {
      int separatore = dati.IndexOf(":", StringComparison.Ordinal);
      Comando = dati.Substring(0, separatore);
      Contenuto = dati.Substring(Comando.Length + 1);
    }
  }
}
```

Servizio.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  /* Classe astratta */
  abstract class Servizio
```

```
  {
    /* Metodo astratto per fornire un determinato servizio */
    abstract public void forniscisiServizio(Ricettore oggetto, List<string> stringa);
  }
}
```

Amicizia.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Amicizia : Servizio
  {
    /* Servizio per aggiungere un nuovo amico */
    override public void forniscisiServizio(Ricettore client, List<string> lista)
    {
      /* Ricerca del ricettore a cui inviare l'amicizia *
        * e creazione della Pending */
      lock (Database.Istanza)
      {
        int esito;

        /* Se l'inserimento dell'amicizia va a buon fine */
        if((esito = Database.Istanza.inserisciAmicizia(lista)) != -1)
        {
          /* Avvisa il Client richiedente che e' andato tutto a buon fine */
          Pacchetto messaggio = new Pacchetto("AggiungiAmico", "OK");
          client.InviaPacchetto(messaggio);

          /* Controlla se il destinatario puo' essere avvisato Online */
          Ricettore destinatario = Connessione.Istanza.cercaInDizionario(lista[1]);

          /* Se e' Online */
          if (destinatario != null)
          {
            /* Se non e' un aggiornamento */
            if (esito == 1)
            {
              /* Inoltra la richiesta al destinatario */
              messaggio = new Pacchetto("InoltroAmicizia", "");
              destinatario.InviaPacchetto(messaggio);
            }
          }
        }
        else
        {
          /* Non e' stato possibile aggiungere l'amico*/
          Pacchetto messaggio = new Pacchetto("AggiungiAmico", "Non e' stato possibile aggiungere
l'amico richiesto");
          client.InviaPacchetto(messaggio);
        }
      }
    }
  }
}
```

Lista.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Lista : Servizio
  {
    /* Servizio per recuperare la lista di amici */
    override public void forniciServizio(Ricettore client, List<string> lista)
    {
      /* Istanza di convertitore */
      Convertitore converti = new Convertitore();
      /* Variabile predisposta per contenere la lista degli amici */
      List<string> listaAmici = new List<string>();

      /* Recupero della lista di amici nel DB */
      lock (Database.Istanza)
        listaAmici = Database.Istanza.listaAmici(lista);

      /* Se la lista non è vuota */
      if (listaAmici.Count != 0)
      {
        /* Invia la lista di amici */
        Pacchetto messaggio = new Pacchetto("ListaAmici", converti.listaAStringa(listaAmici));
        client.InviaPacchetto(messaggio);
      }
      else // altrimenti è vuota
      {
        /* Invia messaggio di lista vuota */
        Pacchetto messaggio = new Pacchetto("ListaAmici", "Non hai ancora alcun amico!");
        client.InviaPacchetto(messaggio);
      }
    }
  }
}
```

Login.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Login : Servizio
  {
    /* Metodo richiamato al momento del Login */
    override public void forniciServizio(Ricettore client, List<string> lista)
    {
      bool esito = false; // Booleano per verifica query
      SQLiteDataReader risultato; // Contenitore risultato Select
      /* Query - lista[0] contiene la username del richiedente - lista[1] contiene la password del
richiedente */
      string query = "SELECT * FROM UTENTI WHERE username = '" + lista[0] + "' AND password = '" +
lista[1] + "'";

      /* Richiesta di Appropriazione dell'istanza della classe Database */
      lock (Database.Istanza)
          risultato = Database.Istanza.cerca(query); // Esecuzione e risultato della Query

      /* Verifica che la query abbia dato risultati */
      while (risultato.Read())
          esito = true;

      /* Se la query ha dato risultati */
      if (esito)
      {
```

```
        /* Aggiorno il dizionario della connessione */
        esito = Connessione.Istanza.utenteOnline(client, lista[0]);

        if (esito)
        {
          /* Avviso l'utente che il Login e' confermato */
          Pacchetto messaggio = new Pacchetto("Login", "OK");
          client.InviaPacchetto(messaggio);
        }
        else
        {
          /* Invia pacchetto contenente il messaggio di errore al Client */
          Pacchetto messaggio = new Pacchetto("Login", "Il tuo username e' gia' loggato in un altra
posizione");
          client.InviaPacchetto(messaggio);
        }
      }
      else // altrimenti, in tutti gli altri casi
      {
        /* Invia pacchetto contenente il messaggio di errore al Client */
        Pacchetto messaggio = new Pacchetto("Login", "I dati inseriti non sono corretti oppure non sei
ancora registrato");
        client.InviaPacchetto(messaggio);
      }
    }
  }
}
```

Messaggistica.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Messaggistica : Servizio
  {
    override public void forniscisiServizio(Ricettore client, List<string> lista)
    {
      /* Lista utilizzata per memorizzare la lista di amici del mittente */
      List<string> listaAmici = new List<string>();

      /* Appropriazione istanza del Database */
      lock (Database.Istanza)
        /* Estraggo la lista di Amici associata al Client */
        listaAmici = Database.Istanza.listaAmici(lista);

      /* Se nella lista degli Amici e' presente l'amico a cui si vuole inviare il messaggio */
      if (listaAmici.Contains(lista[1]))
      {
        /* Confermo l'invio del messaggio al mittente */
        Pacchetto messaggio = new Pacchetto("Messaggio", "OK");
        client.InviaPacchetto(messaggio);

        /* Variabile per la memorizzazione del client destinatario a cui inviare il messaggio */
        Ricettore destinatario;

        /* Ricerca se il Destinatario e' Online */
        lock (Connessione.Istanza)
          destinatario = Connessione.Istanza.cercaInDizionario(lista[1]);

        /* Se il destinatario e' online */
        if (destinatario != null)
        {
          /* Inoltra il messaggio al destinatario immediatamente */
          messaggio = new Pacchetto("InoltroMsg", lista[0] + "," + lista[2] + "," + lista[1]);
          destinatario.InviaPacchetto(messaggio);
```

```
        }
        else // altrimenti
        {
          /* Inserisci il messaggio nella tabella Pending del DB */
          lock (Database.Istanza)
            Database.Istanza.inserisciMessaggio(lista);
        }
      }
      else // Altrimenti, non e' fra gli amici
      {
        /* Invio l'errore relativo al Client */
        Pacchetto messaggio = new Pacchetto("Messaggio", "Impossibile inviare il messaggio. L'utente
non e' tuo amico");
        client.InviaPacchetto(messaggio);
      }
    }
  }
}
```

Registrazione.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Registrazione : Servizio
  {
    /* Metodo richiamato al momento della Registrazione */
    override public void forniciServizio(Ricettore client, List<string> lista)
    {
      bool risultato; // Risultato dell'inserimento dell'utente nel DataBase

      /* Query */
      string query = "INSERT INTO UTENTI (username, password) values ('" + lista[0] + "', '" + lista[1]
+ "')";

      /* Appropriazione dell'istanza del DataBase */
      lock (Database.Istanza)
        risultato = Database.Istanza.inserisci(query); // Inserimento e Risultato della query

      /* Se l'inserimento e' avvenuto con successo */
      if (risultato)
      {
        /* Messaggio positivo */
        Pacchetto messaggio = new Pacchetto("Registrazione", "OK");
        client.InviaPacchetto(messaggio);
      }
      else // altrimenti
      {
        /* Messaggio negativo */
        Pacchetto messaggio = new Pacchetto("Registrazione", "La Username immessa non e' disponibile");
        client.InviaPacchetto(messaggio);
      }
    }
  }
}
```

Segreteria.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class Segreteria : Servizio
  {
    /* Servizio di Segreteria */
    public override void forniscisiServizio(Ricettore client, List<string> lista)
    {
      SQLiteDataReader reader; // Variabile per la lettura di query
      List<string> pending = new List<string>(); // Lista per rendere leggibile il risultato della
query

      /* Query - lista[0] contiene la username del richiedente del servizio */
      string query = "SELECT username, idMessaggio, idAmicizia FROM PENDING WHERE username = '" +
lista[0] + "'";

      /* Ricerca della pending */
      lock (Database.Istanza)
          reader = Database.Istanza.cerca(query);

      /* Se presente la pending, memorizza i dati relativi nella lista */
      if (reader.Read())
      {
        pending.Add(reader["username"].ToString());
        pending.Add(reader["idMessaggio"].ToString());
        pending.Add(reader["idAmicizia"].ToString());
      }

      /* Se la lista non e' vuota */
      if(pending.Count != 0)
      {
        /* Amicizia in Pending (perchè idmessaggio è vuoto) */
        if ((pending[1].Length == 0) && pending[2].Length != 0)
        {
          string richiedente;

          /* Prelievo dell'username che ha chiesto l'amicizia - pending[2] contiene l'id dell'amicizia
*/
          lock (Database.Istanza)
              richiedente = Database.Istanza.prelevaAmicizia(pending[2],lista[0]);

          /* Invio della pending al client */
          Pacchetto messaggio = new Pacchetto("Segreteria", richiedente + " vuole essere tuo amico");
          client.InviaPacchetto(messaggio);

          /* Rimozione della pending */
          Database.Istanza.rimuovi("DELETE FROM PENDING WHERE idAmicizia = '" + pending[2] + "'");
        }

        /* Messaggio in Pending - (perchè idamicizia è vuoto) */
        if (pending[1].Length != 0 && pending[2].Length == 0)
        {
          List<string> listaMessaggio = new List<string>();
          /* Query prelievo messaggio - pending[1] contiene l'id del messaggio */
          query = "SELECT * FROM MESSAGGI WHERE id = '" + pending[1] + "'";

          /* Prelievo del messaggio */
          lock (Database.Istanza)
              reader = Database.Istanza.cerca(query);

          /* Inserimento dei dati relativi in una lista */
          if (reader.Read())
          {
            listaMessaggio.Add(reader["mittente"].ToString());
            listaMessaggio.Add(reader["messaggio"].ToString());
            listaMessaggio.Add(lista[0]);
          }

          /* Inoltra il messaggio al destinatario */
          Pacchetto messaggio = new Pacchetto("Segreteria", "[" + listaMessaggio[0] + "] " +
listaMessaggio[1]);
```

```
          client.InviaPacchetto(messaggio);

          /* Rimozione della pending */
          Database.Istanza.rimuovi("DELETE FROM PENDING WHERE idMessaggio = '" + pending[1] + "'");
        }
      }
      else // altrimenti e' vuota
      {
        /* Invio messaggio segreteria vuota al client */
        Pacchetto messaggio = new Pacchetto("Segreteria", lista[0] + ", la tua segreteria è vuota");
        client.InviaPacchetto(messaggio);
      }
    }
  }
}
```

Database.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Server
{
  class Database
  {
    /* VARIABILI */
    private static Database istanza; // Istanza Singleton Database
    private SQLiteConnection connessione; // Istanza della Connessione
    private int idMessaggi; // ID utilizzato per i record dei messaggi
    private int idAmicizie; // ID utilizzato per i record delle amicizie

    /* COSTRUTTORE */
    private Database()
    {
      /* Creazione DB se non esistente */
      if (!File.Exists("database.sqlite"))
      {
        try
        {
          SQLiteConnection.CreateFile("database.sqlite");
        }
        catch (Exception)
        {
          Interfaccia.stampaMessaggio(" <!> Impossibile creare il Database <!>\n");
          Interfaccia.stampaMessaggio(" <!> L'applicazione ha i permessi per farlo? <!>\n");
          Interfaccia.stampaMessaggio(" -> Chiusura automatica in 10 secondi");
          Thread.Sleep(10000);
          Environment.Exit(0);
        }
        /* Variabile per stabilire la connessione */
        connessione = new SQLiteConnection("Data Source = database.sqlite; Version=3;");
        /* Apertura connessione */
        connessione.Open();
        /* Generazione delle tabelle del DB */
        generaTabella("CREATE TABLE UTENTI (username VARCHAR(24) PRIMARY KEY, password VARCHAR(24))");
        generaTabella("CREATE TABLE MESSAGGI (id INT PRIMARY KEY, mittente VARCHAR(24), messaggio
VARCHAR(250))");
        generaTabella("CREATE TABLE AMICIZIE (id INT PRIMARY KEY, username1 VARCHAR(24), username2
VARCHAR(24), confermaU1 INT, confermaU2 INT)");
        generaTabella("CREATE TABLE PENDING (username VARCHAR(24), idMessaggio INT, idAmicizia INT,
FOREIGN KEY (idMessaggio) REFERENCES MESSAGGI(id), FOREIGN KEY (idAmicizia) REFERENCES AMICIZIE(id)
)");
        /* Inserimento dell'User di Default *
          * USERNAME : Amministratore        *
```

```
        * PASSWORD : Password              */
      inserisci("INSERT INTO UTENTI (username, password) VALUES ('Amministratore','Password')");
    }

    /* Variabile per stabilire la connessione */
    connessione = new SQLiteConnection("Data Source = database.sqlite; Version=3;");
    /* Apertura connessione */
    connessione.Open();

    /* Impostazione del valore iniziale di idAmicizia */
    SQLiteDataReader reader = cerca("SELECT MAX(id) AS max FROM AMICIZIE");
    reader.Read();
    if (!DBNull.Value.Equals(reader["max"]))
      idAmicizie = Convert.ToInt32(reader["max"]);
    else
      idAmicizie = 0;

    /* Impostazione del valore iniziale di idMessaggi */
    reader = cerca("SELECT MAX(id) AS max FROM MESSAGGI");
    reader.Read();
    if (!DBNull.Value.Equals(reader["max"]))
      idMessaggi = Convert.ToInt32(reader["max"]);
    else
      idMessaggi = 0;
  }

  /* METODI */
  /* Metodo per creazione dell'istanza Singleton*/
  public static Database Istanza
  {
    get
    {
      if (istanza == null)
      {
        istanza = new Database();
      }
        return istanza;
    }
  }

  /* Crea la tabella Utente - | ID | Username | Password | */
  private void generaTabella(string query)
  {
    /* Ordine da interpretare */
    SQLiteCommand comando = new SQLiteCommand(query, connessione);
    /* Esecuzione dell'ordine */
    comando.ExecuteNonQuery();
  }

  /* Inserisce i dati nel DB sulla base di una query */
  public bool inserisci(string query)
  {
    try
    {
      /* Prova l'inserimento, in caso di errore ritorna false (eccezione) */
      SQLiteCommand comando = new SQLiteCommand(query, connessione);
      comando.ExecuteNonQuery();
    }
    catch (Exception)
    {
      return false;
    }

    return true;
  }

  /* Cerca i dati nel DB sulla base di una query */
  public SQLiteDataReader cerca(string query)
  {
    /* Esegui la ricerca */
    SQLiteCommand comando = new SQLiteCommand(query, connessione);
    SQLiteDataReader reader = comando.ExecuteReader();

    /* Ritorna il risultato (vuoto in caso di nessuna corrispondenza) */
```

```
      return reader;
    }

    /* Aggiorna i dati nel DB sulla base di una query */
    public int aggiorna(string query)
    {
      int righeAggiornate;

      try
      {
        /* Prova l'aggiornamento, in caso di eccezione segnala 0 righe aggiornate */
        SQLiteCommand comando = new SQLiteCommand(query, connessione);
        righeAggiornate = comando.ExecuteNonQuery();
      }
      catch (Exception)
      {
        righeAggiornate = 0;
      }

      /* Ritorna le righe aggiornate */
      return righeAggiornate;
    }

    /* Rimuovi dati da DB sulla base di una query */
    public bool rimuovi(string query)
    {
      try
      {
        /* Prova la rimozione, in caso di errore ritorna falso */
        SQLiteCommand comando = new SQLiteCommand(query, connessione);
        comando.ExecuteNonQuery();
      }
      catch (Exception)
      {
        return false;
      }

      return true;
    }

    /* Inserimento agile di una Pending Amicizia nel Database */
    public int inserisciAmicizia(List<string> dati)
    {
      /* Esito : -1 => Fallito l'inserimento *
       * Esito :  0 => Amicizia aggiornata   *
       * Esito :  1 => Amicizia inserita     */
      int esito;

      /* Controllo se l'amico da voler aggiungere esiste */
      string query = "SELECT * FROM UTENTI WHERE username = '" + dati[1] + "'";
      SQLiteDataReader reader = cerca(query);

      /* Se esiste */
      if(reader.Read())
      {
        /* Controllo che esista gia' l'amicizia */
        query = "SELECT * FROM AMICIZIE WHERE username1 = '" + dati[0] + "' AND username2 = '" +
dati[1] + "'";
        reader = cerca(query);

        /* Se esiste */
        if (reader.Read())
        {
          /* Se risulta che l'amicizia non sia gia' instaurata, la instauro */
          if (reader["confermaU1"].ToString() != "1")
          {
            query = "UPDATE AMICIZIE SET confermaU1 = 1 WHERE id = '" + reader["id"].ToString() + "'";
            aggiorna(query);
            esito = 0;
          }
          else
            esito = -1;
        }
        else //altrimenti, non esiste, ma va verificato il reciproco
```

```
        {
          /* Controllo che esista gia' l'amicizia a ruoli invertiti */
          query = "SELECT * FROM AMICIZIE WHERE username1 = '" + dati[1] + "' AND username2 = '" +
dati[0] + "'";
          reader = cerca(query);

          /* Se l'amicizia esiste */
          if (reader.Read())
          {
            /* Se risulta che l'amicizia non sia gia' instaurata, la instauro */
            if (reader["confermaU2"].ToString() != "1")
            {
              query = "UPDATE AMICIZIE SET confermaU2 = 1 WHERE id = '" + reader["id"].ToString() +
"'";
              aggiorna(query);
              esito = 0;
            }
            else
              esito = -1;
          }
          else // altrimenti, non esiste, e' possibile aggiungerla
          {
            /* Aggiorno l'id corrente */
            idAmicizie++;

            /* Tentativo di inserimento */
            query = "INSERT INTO AMICIZIE (id, username1, username2, confermaU1, confermaU2) VALUES ('"
+ idAmicizie + "','" + dati[0] + "', '" + dati[1] + "', 1, 2)";
            bool inserito = inserisci(query);

            /* Se l'inserimento è andato a buon fine */
            if (inserito)
            {
              /* Tenta l'inserimento della pending */
              query = "INSERT INTO PENDING (username, idMessaggio, idAmicizia) VALUES ('" + dati[1] +
"', null, '" + idAmicizie + "')";
              inserito = inserisci(query);

              /* Se l'inserimento va a buon fine */
              if (inserito)
                esito = 1;
              else // altrimenti
              {
                /* Rimuovi l'amicizia inserita in precedenza */
                query = "DELETE FROM AMICIZIE WHERE id = '" + idAmicizie + "'";
                rimuovi(query);

                /* Riporta l'idAmicizie allo stato precedente */
                if (idAmicizie > 0)
                    idAmicizie--;
                esito = -1;
              }
            }
            else // altrimenti non è andato a buon fine
            {
              /* Riporta idAmicizie allo stato precedente */
              if (idAmicizie > 0)
                  idAmicizie--;
              esito = -1;
            }
          }
        }
      }
      else
          esito = -1;

      return esito;
    }

    /* Inserimento agile di una Pending Messaggio nel Database */
    public int inserisciMessaggio(List<string> dati)
    {
      /* Esito : -1 => Fallito l'inserimento *
       * Esito :  1 => Messaggio Inserito    */
```

```
      int esito;

      /* Controllo se l'amico a cui voler inviare il messaggio esiste */
      string query = "SELECT * FROM UTENTI WHERE username = '" + dati[1] + "'";
      SQLiteDataReader reader = cerca(query);

      /* Se esiste */
      if (reader.Read())
      {
        /* Aggiornamento variabile idMessaggi */
        idMessaggi++;

        query = "INSERT INTO MESSAGGI  (id, mittente, messaggio) VALUES ('" + idMessaggi + "','" +
dati[0] + "', '" + dati[2] + "')";

        /* Se l'inserimento del messaggio va a buon fine */
        if (inserisci(query))
        {
          query = "INSERT INTO PENDING (username, idMessaggio, idAmicizia) VALUES ('" + dati[1] + "','"
+ idMessaggi + "', null)";

          /* Prova a inserire anche la pending */
          if (inserisci(query))
              esito = 1;
          else // a;trimenti si è verificato un errore
          {
            /* Rimuovi il messaggio precedentemente inserito */
            query = "DELETE FROM MESSAGGI WHERE id = '" + idMessaggi + "'";
            rimuovi(query);

            /* Riporta l'idMessaggi allo stato precedente */
            if (idMessaggi > 0)
                idMessaggi--;
            esito = -1;
          }
        }
        else // altrimenti, si è verificato un errore
        {
          /* Quindi porta idMessaggi allo stato precedente */
          if (idMessaggi > 0)
              idMessaggi--;
          esito = -1;
        }
      }
      else // altrimenti
        esito = -1;

      /* Ritorna l'esito del metodo */
      return esito;
    }

    /* Metodo per la restituzione della lista di amici di un determinato utente */
    public List<string> listaAmici(List<string> datiUtente)
    {
      /* Lista nella quale saranno posizionati gli amici */
      List<string> lista = new List<string>();

      /* Restituisci le amicizie dove username1 corrisponde allo username del richiedente */
      string query = "SELECT username2 FROM AMICIZIE WHERE username1 = '" + datiUtente[0] + "' AND
confermaU1 = 1 AND confermaU2 = 1";
      SQLiteDataReader reader = cerca(query);

      /* Se c'è corrispondenza, inserisci gli elementi trovati nella lista */
      while (reader.Read())
          lista.Add(reader["username2"].ToString());

      /* Restituisci le amicizie dove username2 corrisponde allo username del richiedente */
      query = "SELECT username1 FROM AMICIZIE WHERE username2 = '" + datiUtente[0] + "' AND confermaU1
= 1 AND confermaU2 = 1";
      reader = cerca(query);

      /* Se c'è corrispondenza, inserisci gli elementi trovati nella lista */
      while (reader.Read())
          lista.Add(reader["username1"].ToString());
```

```
      /* Ritorna la lista */
      return lista;
    }

    /* Metodo per il prelievo di una amicizia dal DB tramite id */
    public string prelevaAmicizia(string id, string username)
    {
      /* Ricerca l'amicizia */
      string query = "SELECT username1, username2 FROM AMICIZIE WHERE id = '" + id + "'";
      SQLiteDataReader reader = cerca(query);

      /* Leggi gli utenti coinvolti */
      string utente1 = reader["username1"].ToString();
      string utente2 = reader["username2"].ToString();

      /* Restituisci l'utente non richidente */
      if (username == utente1)
          return utente2;

      if (username == utente2)
          return utente1;

      /* Se non c'è alcuna corrispondenza con il richiedente, null */
      return null;
    }
  }
}
```

CLIENT

Interfaccia.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Text.RegularExpressions;

namespace Client
{
  static class Interfaccia
  {
    /* VARIABILI */
    /* Istanza della classe collegamento */
    static Connessione collegamento;
    /* Istanza della classe richiesta */
    static Richiesta richiesta = new Richiesta();
    /* Mutex per la corretta sincronizzazione chiamate sincrone */
    static Semaphore mutex = new Semaphore(0,1);

    /* Avvio per Argomenti                        *
     * Argomento mancante - Avvio Normale         *
     * Argomento presente - Errore                */
    static void Main(string[] args)
    {
      /* Testo di Apertura */
      testoApertura();

      /* Controllo del numero di argomenti */
      if (args.Length < 1)
      {
        /* Stabilimento della connessione con il Server */
        stabilisciConnessione("127.0.0.1", 5302);

        /* Lancia il Servizio di Login e Registrazione */
        login();
      }
      else
      {
        /* Messaggio d'errore */
        Console.WriteLine(" <!> Hai inserito argomenti non necessari <!>");
        Console.WriteLine(" <!> Premi Invio e riavvia l'applicazione senza argomenti <!>");
        Console.ReadLine();
      }

      /* Chiusura ambiente */
      Environment.Exit(0);
    }

    /* METODI */
    /* Metodo per la stampa del messaggio di Benvenuto a schermo */
    static void testoApertura()
    {
      Console.WriteLine("\t\t  * * * * * * * * * * * * * * * * * * * * *");
      Console.WriteLine("\t\t  *   Progetto di Ingegneria del Software   *");
      Console.WriteLine("\t\t  *    Software Messaggistica Istantanea    *");
      Console.WriteLine("\t\t  *    Bigini Gioele - Matricola: 261990    *");
      Console.WriteLine("\t\t  *          Versione Software: 1.0          *");
      Console.WriteLine("\t\t  * * * * * * * * * * * * * * * * * * * * *\n\n");
    }

    /* Metodo per la stampa del messaggio di Connessione al Server */
    static void stabilisciConnessione(string indirizzoIP, int porta)
    {
      try
```

```
    {
      Console.WriteLine(" -> Tento di stabilire la connessione con il Server\n");

      /* Istanziazione dell'oggetto Connessione (Singleton) */
      collegamento = Connessione.Istanza;

      Console.WriteLine(" <> Connessione stabilita\n\n");
    }
    catch (Exception)
    {
      Console.WriteLine(" (!) Impossibile connettersi al Server. Il Server è OFFLINE\n");
      Console.WriteLine(" -> Premere Invio per chiudere l'applicazione");
      Console.ReadLine();
      Environment.Exit(0);
    }
}

/* Metodo per la stampa dei Servizi di Accesso ai Servizi veri e propri */
static void testoLogin()
{
  Console.WriteLine("\t\t       *  Unisciti ora alla Community  *\n\n");
  Console.WriteLine(" 1. Effettua il Login");
  Console.WriteLine(" 2. Registrati ora!");
  Console.WriteLine(" 3. Chiudi l'Applicazione\n");
}

/* Metodo per la stampa dei Servizi di Accesso ai Servizi veri e propri *
  * I Servizi sono:                                                     *
  * - Login                                                             *
  * - Registrazione                                                     */
static void login()
{
  string username, // Stringa di memorizzazione dello Username dell'utente
         password; // Stringa di memorizzazione della Password dell'utente

  testoLogin(); // Stampa a schermo dei Servizi di Accesso

  /* Variabile che descrive i tasti premuti sulla tastiera */
  ConsoleKeyInfo pressione = new ConsoleKeyInfo();

  /* Ciclo che resta in attesa della pressione dei tasti [1,2,3] *
    * 1. Login                                                   *
    * 2. Registrazione                                           *
    * 3. Chiudi Applicazione                                     */
  while (pressione.Key != ConsoleKey.D1 &&
         pressione.Key != ConsoleKey.D2 &&
         pressione.Key != ConsoleKey.D3)
  {
    pressione = Console.ReadKey(true);

    switch (pressione.Key)
    {
        /* 1. Login */
        case ConsoleKey.D1:
            Console.Clear();
            /* Acquisizione dati da Utente */
            testoApertura();
            Console.WriteLine(" -> Tento di stabilire la connessione con il Server\n");
            Console.WriteLine(" <> Connessione stabilita\n\n");
            testoLogin();
            Console.WriteLine("\t\t\t\t  * Login *\n");
            Console.Write(" Inserisci la tua Username: ");
            username = Console.ReadLine();
            Console.Write(" Inserisci la tua Password: ");
            password = Console.ReadLine();

            /* Se l'inserimento rispetta le seguenti regole *
              * 1. Limite 24 Caratteri                      *
              * 2. Solo lettere maiuscole e minuscole       */
            if (Regex.IsMatch(username, @"^[a-zA-Z]+$") &&
                Regex.IsMatch(password, @"^[a-zA-Z]+$") &&
                username.Length <= 24 &&
                password.Length <= 24)
            {
```

```
                /* Chiama il Servizio */
                richiesta.login(collegamento, username, password);

                /* Attendi la risposta dal Server */
                Console.WriteLine("\n -> Verifica delle credenziali di accesso\n");
                mutex.WaitOne();

                /* Se la risposta del Server non e' OK */
                if (collegamento.chiamataSincrona != "OK")
                {
                  /* Messaggio di Errore dal Server */
                  Console.WriteLine(" [ERRORE] {0}\n", collegamento.chiamataSincrona.ToString());
                  pressione = new ConsoleKeyInfo();
                }
                else
                    servizi(username); // Avvia i Servizi per il Client
              }
              else
              {
                /* Messaggio di Errore */
                Console.WriteLine("\n (!) Sono consentiti non piu' di 24 caratteri minuscoli e
maiuscoli");

                /* Reset della pressione */
                pressione = new ConsoleKeyInfo();
              }
              break;

          /* 2. Registrazione */
          case ConsoleKey.D2:
              Console.Clear();
              testoApertura();
              Console.WriteLine(" -> Tento di stabilire la connessione con il Server\n");
              Console.WriteLine(" <> Connessione stabilita\n\n");
              testoLogin();
              /* Acquisizione dati da Utente */
              Console.WriteLine("\t\t\t      *  Registrazione *\n");
              Console.Write(" Inserisci la tua Username: ");
              username = Console.ReadLine();
              Console.Write(" Inserisci la tua Password: ");
              password = Console.ReadLine();

              /* Se l'inserimento (username e password) rispetta le seguenti regole: *
                * 1. Limite 24 Caratteri                                              *
                * 2. Solo lettere maiuscole e minuscole                               */
              if (Regex.IsMatch(username, @"^[a-zA-Z]+$") &&
                  Regex.IsMatch(password, @"^[a-zA-Z]+$") &&
                  username.Length <= 24 &&
                  password.Length <= 24)
              {
                /* Chiama il Servizio */
                richiesta.registrazione(collegamento, username, password);

                /* Attendi la risposta dal Server */
                Console.WriteLine("\n -> Verifica delle credenziali di accesso\n");
                mutex.WaitOne();

                /* Se la risposta del Server non e' OK */
                if (collegamento.chiamataSincrona != "OK")
                    /* Messaggio di errore dal Server */
                    Console.WriteLine(" [ERRORE] {0}\n", collegamento.chiamataSincrona.ToString());
                else
                    /* Messaggio registrazione avvenuta */
                    Console.WriteLine(" [OK] Sei registrato. Ora puoi effettuare il Login\n");
              }
              else
                /* Messaggio di Errore */
                Console.WriteLine("\n (!) Sono consentiti non piu' di 24 caratteri minuscoli e
maiuscoli");

              /* Reset della pressione */
              pressione = new ConsoleKeyInfo();
              break;
```

```
            /* 3. Chiusura del Programma */
            case ConsoleKey.D3:
                break;

            /* Altra Pressione */
            default:
                Console.Clear();
                testoApertura();
                Console.WriteLine(" -> Tento di stabilire la connessione con il Server\n");
                Console.WriteLine(" <> Connessione stabilita\n\n");
                testoLogin();
                Console.WriteLine(" (!) Errore nella scelta del Servizio\n");
                break;
        }
      }
    }

    /* Metodo per la stampa dei Servizi di Accesso ai Servizi veri e propri */
    static void testoServizi(string username)
    {
      testoApertura();
      Console.WriteLine("\t\t\t\t  * Servizi *\n");
      Console.WriteLine(" [{0}]\n", username);
      Console.WriteLine(" 1. Segreteria");
      Console.WriteLine(" 2. Invia un messaggio");
      Console.WriteLine(" 3. Ottieni la lista dei tuoi amici");
      Console.WriteLine(" 4. Aggiungi un Amico");
      Console.WriteLine(" 5. Chiudi l'Applicazione\n");
    }

    /* Metodo per la stampa dei Servizi in seguito al Login *
     * I Servizi sono:                                      *
     * - Segreteria                                         *
     * - Invia un messaggio                                 *
     * - Ottieni la lista dei tuoi amici                    *
     * - Aggiungi un amico                                  */
    static void servizi(string username)
    {
```

Le righe di codice successive creano un Thread che si occupa di mostrare i messaggi asincroni provenienti dal Server. Il flusso di esecuzione viene tenuto attivo quando l'Utente è posizionato nella schermata iniziale dei Servizi e cancellato nel momento in cui si usufruisce di un servizio. Con questo sistema si evita la visualizzazione di messaggi asincroni in momenti sbagliati.

```
      /* Thread che si occupa di stampare a video le chiamate asincrone *
       * Esso viene interrotto nel momento in cui un utente accede ad   *
       * un servizio e riattivato solo in seguito                       */
      Thread controllaCoda = new Thread(stampaChiamataAsincrona);
      controllaCoda.Start();

      /* Messaggio a schermo dei Servizi */
      Console.Clear();
      testoServizi(username);

      /* Variabile che descrive i tasti premuti sulla tastiera */
      ConsoleKeyInfo pressione = new ConsoleKeyInfo();

      /* Istanza di Convertitore per convertire le stringhe */
      Convertitore converti = new Convertitore();
      /* Lista per il contenuto convertito in liste */
      List<string> lista;

      /* Ciclo che resta in attesa della pressione dei tasti [1,2,3,4,5] *
       * 1. Segreteria                                                   *
       * 2. Invia un Messaggio                                           *
       * 3. Lista degli Amici                                            *
       * 4. Aggiungi un amico                                            *
       * 5. Chiudi Applicazione                                          */
      while (pressione.Key != ConsoleKey.D1 &&
             pressione.Key != ConsoleKey.D2 &&
             pressione.Key != ConsoleKey.D3 &&
             pressione.Key != ConsoleKey.D4 &&
             pressione.Key != ConsoleKey.D5 &&
```

```
            pressione.Key != ConsoleKey.D6 &&
            pressione.Key != ConsoleKey.D7)
      {
        pressione = Console.ReadKey(true);

        switch (pressione.Key)
        {
          /* 1. Segreteria */
          case ConsoleKey.D1:
              /* Interruzione del Server */
              controllaCoda.Abort();
              /* Acquisizione dati da utente */
              Console.Clear();
              testoApertura();
              Console.WriteLine("\t\t\t\t* Segreteria *\n");

              do
              {
                /* Invio della richiesta al Server */
                richiesta.segreteria(collegamento,username);

                /* Attesa della riposta da parte del Server */
                Console.WriteLine(" <> Prelievo dei dati in Segreteria...\n");
                mutex.WaitOne();

                /* Scrivi la risposta a schermo */
                Console.WriteLine(" -> {0}", Connessione.Istanza.chiamataSincrona);

                Console.WriteLine("\n Continuo a verificare?S/N\n");

                /* Variabile che descrive i tasti premuti sulla tastiera */
                pressione = new ConsoleKeyInfo();

                /* Cattura della pressione */
                while (pressione.Key != ConsoleKey.S && pressione.Key != ConsoleKey.N)
                {
                  pressione = Console.ReadKey(true);

                  switch (pressione.Key)
                  {
                    case ConsoleKey.S:
                        break;
                    case ConsoleKey.N:
                        break;
                  }
                }
              } while(pressione.Key == ConsoleKey.S);

              /* Messaggio di ritorno ai servizi */
              Console.WriteLine(" <> Premi Invio per tornare ai Servizi\n");
              Console.ReadLine();
              Console.Clear();
              testoServizi(username);

              /* Reset della pressione */
              pressione = new ConsoleKeyInfo();

              /* Riavvio Thread chiamate asincrone */
              controllaCoda = new Thread(stampaChiamataAsincrona);
              controllaCoda.Start();
              break;

          /* 2. Invia un Messaggio */
          case ConsoleKey.D2:
              /* Interruzione del Server */
              controllaCoda.Abort();
              /* Acquisizione dati da utente */
              Console.Clear();
              testoApertura();
              Console.WriteLine(" [SUGGERIMENTO]");
              Console.WriteLine(" Per chattare devi essere amico della persona con cui desideri
farlo!\n\n");
              Console.WriteLine("\t\t\t     * Invia un Messaggio *\n");
              Console.Write(" A chi vuoi inviare il messaggio: ");
```

```
            string destinatario = Console.ReadLine().ToString();
            Console.Write(" Messaggio: ");
            string messaggio = Console.ReadLine().ToString();

            /* Se l'inserimento rispetta le seguenti regole *
              * 1. Limite 24 Caratteri                      *
              * 2. Solo lettere maiuscole e minuscole       *
              * 3. Solo altri utenti (non se stessi)        */
            if (Regex.IsMatch(destinatario, @"^[a-zA-Z]+$") &&
                Regex.IsMatch(messaggio, @"^[^,]+$") &&
                destinatario.Length <= 24 &&
                destinatario != username &&
                messaggio.Length <= 256)
            {
              richiesta.inviaMessaggio(collegamento, username, destinatario, messaggio);

              /* Attesa della riposta da parte del Server */
              Console.WriteLine("\n -> Attendere prego\n");
              mutex.WaitOne();

              /* Risposta del Server */
              if (collegamento.chiamataSincrona != "OK")
                  Console.WriteLine(" [ERRORE] {0}\n", collegamento.chiamataSincrona);
              else
                  Console.WriteLine(" [OK] Messaggio Inviato\n");
            }
            else
            {
              /* Messaggio di Errore */
              Console.WriteLine("\n [MESSAGGIO NON INVIATO]");
              Console.WriteLine(" (!) Non puoi scrivere a te stesso");
              Console.WriteLine(" (!) In questa versione la virgola non e' ammessa\n");
            }

            /* Messaggio di ritorno ai Servizi */
            Console.WriteLine(" <> Premi Invio per tornare ai Servizi\n");
            Console.ReadLine();
            Console.Clear();
            testoServizi(username);

            /* Reset della pressione */
            pressione = new ConsoleKeyInfo();

            /* Riavvio Thread chiamate asincrone */
            controllaCoda = new Thread(stampaChiamataAsincrona);
            controllaCoda.Start();
            break;

        /* 3. Lista degli Amici */
        case ConsoleKey.D3:
            /* Interruzione del Server */
            controllaCoda.Abort();
            /* Acquisizione dati da utente */
            Console.Clear();
            testoApertura();
            Console.WriteLine("\t\t\t       * Lista Amicizie *\n", username);
            /* Invia la richiesta */
            richiesta.listaAmici(collegamento,username);

            /* Attendi la risposta del Server */
            Console.WriteLine(" <> Recupero della lista di amici...\n");
            mutex.WaitOne();

            try
            {
                /* Scrivi la risposta a schermo */
                lista = converti.StringaALista(Connessione.Istanza.chiamataSincrona);

                foreach (string amico in lista)
                    Console.WriteLine(" -> {0}", amico);
            }
            catch (Exception)
            {
                Console.WriteLine(" (!) Si e' verificato un errore nella richiesta. Riprova\n");
```

```
                lista = new List<string>();
            }

            /* Rimanda ai Servizi se l'utente preme invio */
            Console.WriteLine("\n <> Premi Invio per tornare ai Servizi\n");
            Console.ReadLine();
            Console.Clear();
            testoServizi(username);

            /* Reset della pressione */
            pressione = new ConsoleKeyInfo();

            /* Riavvio Thread chiamate asincrone */
            controllaCoda = new Thread(stampaChiamataAsincrona);
            controllaCoda.Start();
            break;

        /* 4. Aggiungi un amico */
        case ConsoleKey.D4:
            /* Interruzione del Server */
            controllaCoda.Abort();
            /* Acquisizione dati da utente */
            Console.Clear();
            testoApertura();
            Console.WriteLine(" [SUGGERIMENTO]");
            Console.WriteLine(" Se sei stato aggiunto, per confermare l'amicizia e' sufficiente che
anche");
            Console.WriteLine(" tu aggiungi l'Utente!\n\n");
            Console.WriteLine("\t\t\t     * Aggiungi un amico *\n");
            Console.Write(" Username dell'amico: ");
            string amicoAggiunto = Console.ReadLine();

            /* Se l'inserimento rispetta le seguenti regole *
              * 1. Limite 24 Caratteri                      *
              * 2. Solo lettere maiuscole e minuscole       *
              * 3. Solo altri utenti (non se stessi)        */
            if (Regex.IsMatch(amicoAggiunto, @"^[a-zA-Z]+$") && amicoAggiunto.Length <= 24 &&
                amicoAggiunto != username)
            {
              richiesta.aggiungiAmico(collegamento, username, amicoAggiunto);
              Console.WriteLine();

              /* Attesa della risposta dal Server */
              Console.WriteLine(" -> Attendere prego\n");
              mutex.WaitOne();

              /* Se la risposta dal Server non e' OK */
              if (collegamento.chiamataSincrona != "OK")
                  /* Comunicazione dell'errore del Server */
                  Console.WriteLine(" [ERRORE] {0}\n", collegamento.chiamataSincrona);
              else // altrimenti
                  /* Conferma che il servizio richiesto e' andato a buon fine */
                  Console.WriteLine(" [OK] Amico aggiunto. Contattalo se lo vedi nella tua lista di
amici\n");
            }
            else // altrimenti
            {
              /* Messaggio di Errore */
              Console.WriteLine("\n [AMICO NON AGGIUNTO]");
              Console.WriteLine(" (!) Non puoi aggiungere te stesso");
              Console.WriteLine(" (!) Sono ammessi caratteri alfabetici maiuscoli e minuscoli\n");
            }

            /* Ritorna alla schermata dei Servizi */
            Console.WriteLine(" <> Premi Invio per tornare ai Servizi\n");
            Console.ReadLine();
            Console.Clear();
            testoServizi(username);

            /* Reset della pressione */
            pressione = new ConsoleKeyInfo();

            /* Riavvio Thread chiamate asincrone */
            controllaCoda = new Thread(stampaChiamataAsincrona);
```

```
            controllaCoda.Start();
            break;

        /* 5. Chiudi Applicazione */
        case ConsoleKey.D7:
            break;

        /* Altra pressione */
        default:
            Console.Clear();
            testoServizi(username);
            Console.WriteLine(" (!) Errore nella scelta del Servizio\n");
            break;
    }
  }
}

/* Metodo chiamato in caso di ricevimento Real-Time di una amicizia */
static void stampaChiamataAsincrona()
{
  Convertitore converti = new Convertitore(); // Istanza di Convertitore
  Pacchetto pacchetto;    // Variabile per chimata asincrona
  bool condizione = true; // Condizione

  /* La condizione salta nel momento in cui viene interrotto il Thread */
  while (condizione)
  {
    try
    {
      try
      {
        /* Estraggo la prima chiamata asincrona memorizzata */
        pacchetto = Connessione.Istanza.chiamateAsincrone.Dequeue();

        /* Messaggio */
        if(pacchetto.Comando == "InoltroMsg")
        {
          /* Conversione della stringa in lista */
          List<string> valori = converti.StringaALista(pacchetto.Contenuto);
          /* Avviso a schermo */
          Console.WriteLine(" [{0}] {1}", valori[0], valori[1]);
        }

        /* Amicizia */
        if (pacchetto.Comando == "InoltroAmicizia")
        {
          /* Avviso a schermo */
          Console.WriteLine(" -> Hai una richiesta di amicizia in sospeso in Segreteria\n");
        }
      }
      catch (InvalidOperationException)
      {
        /* Metti in pausa il Thread per un secondo */
        Thread.Sleep(1000);
      }
    }
    catch (ThreadInterruptedException)
    {
      condizione = false;
    }
  }
}

/* Metodo chiamato in caso di ricevimento di un Errore */
public static void stampaErrore(string errore)
{
  Console.WriteLine("\n <!> {0}", errore);

  /* Errori critici che necessitano della chiusura dell'applicazione */
  if (errore == "Il Server e' OFFLINE" ||
      errore == "Il Server ti ha inviato un pacchetto anomalo")
  {
    for (int i = 20; i > 0; i--)
    {
```

```
            /* Countdown a schermo */
            Console.Clear();
            testoApertura();
            Console.WriteLine("\n <!> {0}. CTRL-C per chiudere l'applicazione immediatamente\n", errore);
            Console.WriteLine(" <!> Spegnimento automatico in {0} secondi\n", i);
            Thread.Sleep(1000);
          }
          /* Chiusura */
          Environment.Exit(0);
        }
      }

      /* Funzione accessibile per garantire la corretta sincronizzazione dei flussi d'esecuzione */
      public static void liberaMutex()
      {
        mutex.Release();
      }
    }
}
```

Connessione.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Client
{
  class Connessione
  {
    /* VARIABILI */
    private static Connessione istanza; // Istanza Singleton
    public TcpClient connessioneTCP { get; private set; } // Connessione TCP
    private NetworkStream canale; // Canale di comunicazione
    private StreamReader lettore; // Lettore del canale
    private StreamWriter scrittore; // Scrittore del canale
    private Thread ascoltaServer; // Thread in ascolto al Server
```

Le chiamate sincrone, essendo tali, necessitano di una variabile nella quale inserire il dato ricevuto dal Server. Le chiamate asincrone, invece, potrebbero arrivare in qualsiasi momento e quindi, accumularsi. Per gestire questo accumulo si utilizza una coda che viene svuotata dal Thread per la stampa delle chiamate asincrone nella classe Interfaccia.

```
    /* Variabile nella quale viene depositata la risposta ricevuta SINCRONA */
    public string chiamataSincrona { get; private set; }
    /* Variabile nelle quali vengono depositate le risposte ricevute ASINCRONE */
    public Queue<Pacchetto> chiamateAsincrone = new Queue<Pacchetto>();

    /* COSTRUTTORE */
    private Connessione()
    {
      /* Creazione della connessione TCP con il Server */
      connessioneTCP = new TcpClient();
```

La seguente riga consente di modificare l'IP e la porta nella quale pescare il Server Remoto.

```
      connessioneTCP.Connect(IPAddress.Parse("127.0.0.1"), 5302);

      /* Assegnamento strumenti di dialogo nel canale stabilito con il Server */
      canale = connessioneTCP.GetStream();
      scrittore = new StreamWriter(canale);
      lettore = new StreamReader(canale);
```

```
    /* Creazione e Avvio di un Thread dedicato al'ascolto del Server */
    ascoltaServer = new Thread(ricezione);
    ascoltaServer.Start();
  }

  /* METODI */
  /* Metodo per creazione dell'istanza Singleton*/
  public static Connessione Istanza
  {
    get
    {
      if (istanza == null)
      {
        istanza = new Connessione();
      }
      return istanza;
    }
  }
```

Il seguente metodo è sempre attivo grazie ad un flusso di esecuzione, come nell'applicazione Client, per verificare continuamente la presenza di dati nel canale di comunicazione.

```
  /* Metodo per la ricezione dei dati nel canale dal Server Remoto */
  private void ricezione()
  {
    bool condizione = true; // Condizione per il ciclo
    List<int> listaCaratteri = new List<int>(); // Lista dei caratteri acquisiti in ricezione
    int valore = new int(); // Variabile per valore acquisito nel canale

    /* Esegui le istruzioni se il Server è connesso                *
     * Interrompi il ciclo in caso di Disconnessione del Server */
    do
    {
      try
      {
        /* Acquisizione del valore */
        valore = lettore.Read();

        /* Se non è il valore terminatore */
        if (valore != 0)
          listaCaratteri.Add(valore); // Aggiunta del valore alla lista
        /* Altrimenti, è il valore terminatore, composizione del messaggio ricevuto */
        else
        {
          var messaggio = new StringBuilder();

          /* Costruzione del messaggio ricevuto - Conversione degli interi in caratteri */
          for (int i = 0; i < listaCaratteri.Count; i++)
              messaggio.Append(Convert.ToChar(listaCaratteri[i]));

          /* Invio del messaggio ricevuto per l'elaborazione */
          elaboraRichiesta(messaggio.ToString());

          /* Svuoto la lista degli interi */
          listaCaratteri.Clear();
        }
      }
      catch (IOException) // Se il Server si disconnette
      {
        /* Esegui una routine */
        Interfaccia.stampaErrore("Il Server e' OFFLINE");
        /* Uscita dal Ciclo */
        condizione = false;
      }
    } while (condizione);
  }

  /* Metodo per l'elaborazione dei messaggi ricevuti dal Server */
  private void elaboraRichiesta(string richiesta)
  {
    /* Generazione di un pacchetto interpretabile */
    Pacchetto pacchetto = new Pacchetto(richiesta);
```

```
      /* Estrazione del comando */
      string comando = pacchetto.Comando;

      /* Comportamento relativo al comando ricevuto */
      switch (comando)
      {
        /* Chiamate sincrone */
        case "Login":
        case "Registrazione":
        case "AggiungiAmico":
        case "Messaggio":
        case "ListaAmici":
        case "Segreteria":
            chiamataSincrona = pacchetto.Contenuto;
            Interfaccia.liberaMutex();
            break;

        /* Chiamate asincrone */
        case "InoltroAmicizia":
        case "InoltroMsg":
            chiamateAsincrone.Enqueue(pacchetto);
            break;

        /* Se il server spedisce pacchetti anomali */
        default:
            Interfaccia.stampaErrore("Il server ti ha inviato un pacchetto anomalo");
            break;
      }
    }

    /* Questo Metodo, con il successivo, permette l'invio dei pacchetti nel NetworkStream */
    public void InviaPacchetto(Pacchetto pacchetto)
    {
      /* Invio nel NetworkStream */
      scrittore.Write("{0}:{1}\0", pacchetto.Comando, pacchetto.Contenuto);
      /* Svuoto lo Stream */
      scrittore.Flush();
    }
  }
}
```

Convertitore.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
  class Convertitore
  {
    /* METODI */
    /* Metodo per la conversione da lista a stringa */
    public string ListaAStringa(List<string> lista)
    {
      string stringaOttenuta; // stringa di ritorno

      /* Controlla che la lista non sia vuota */
      if (lista.Count != 0)
      {
        /* utilizzo un boolean per creare un comportamento diverso sul *
          * primo elemento della lista */
        bool primoElemento = true;

        /* Creo una variabile di tipo StringBuilder */
        var stringa = new StringBuilder();

        /* Costruisco la stringa di questo tipo: *
          * Stringa1,Stringa2,Stringa3...         */
        foreach (var elemento in lista)
```

```
        {
          if (primoElemento)
          {
            stringa.Append(elemento);
            primoElemento = false;
          }
          else
          {
            stringa.Append(string.Format(",{0}", elemento));
          }
        }

        /* StringBuilder non è tipo string ma è facilmente rapportabile */
        stringaOttenuta = stringa.ToString();
      }
      else
      {
        stringaOttenuta = null; // Se vuota il valore è null
      }

      /* Ritorno della stringa */
      return stringaOttenuta;
    }

    /* Metodo per la conversione da Stringa a Lista */
    public List<string> StringaALista(string stringa)
    {
      /* Variabile che ospiterà la lista */
      List<string> lista = new List<string>();

      /* Se la stringa non è null o vuota */
      if (!string.IsNullOrEmpty(stringa))
      {
        /* Prova a comporre la stringa */
        try
        {
          foreach (string elemento in stringa.Split(','))
          {
            lista.Add(elemento);
          }
        }
        catch (Exception)
        {
          lista = null; // in caso di errore invia lista vuota
        }
      }
      else
      {
        lista = null; // La stringa è null, allora la lista è null
      }

      /* Ritorno della lista */
      return lista;
    }
  }
}
```

Pacchetto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
  /* Classe per la generazione dei pacchetti */
  class Pacchetto
  {
    /* VARIABILI */
    /* I pacchetti contengono i parametri                *
```

```
      * Comando   : Nome del servizio richiesto o offerto *
      * Contenuto : Contenuto devoluto                    */
    public string Comando { get; private set; }
    public string Contenuto { get; private set; }

    /* COSTRUTTORI */
    /* 2 Costruttori da utilizzare secondo l'uso */
    /* Costruzione di un pacchetto direttamente con i parametri assegnati */
    public Pacchetto(string comando, string contenuto)
    {
      Comando = comando;
      Contenuto = contenuto;
    }

    /* Costruttore di un pacchetto attraverso una stringa */
    public Pacchetto(string dati)
    {
      int separatore = dati.IndexOf(":", StringComparison.Ordinal);
      Comando = dati.Substring(0, separatore);
      Contenuto = dati.Substring(Comando.Length + 1);
    }
  }
}
```

Richiesta.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Client
{
  /* La classe contiene tutte le richieste eseguibili dal Client */
  class Richiesta
  {
    /* Richiesta di aggiunta di un amico */
    public void aggiungiAmico(Connessione collegamento, string username, string amico)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
        var msgPack = new Pacchetto("AggiungiAmico", string.Format("{0},{1}", username, amico));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }

    /* Richiesta di invio di un messaggio */
    public void inviaMessaggio(Connessione collegamento, string mittente, string destinatario, string
messaggio)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
        var msgPack = new Pacchetto("Messaggio", string.Format("{0},{1},{2}", mittente, destinatario,
messaggio));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }

    /* Richiesta ottenimento lista amici */
    public void listaAmici(Connessione collegamento, string username)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
```

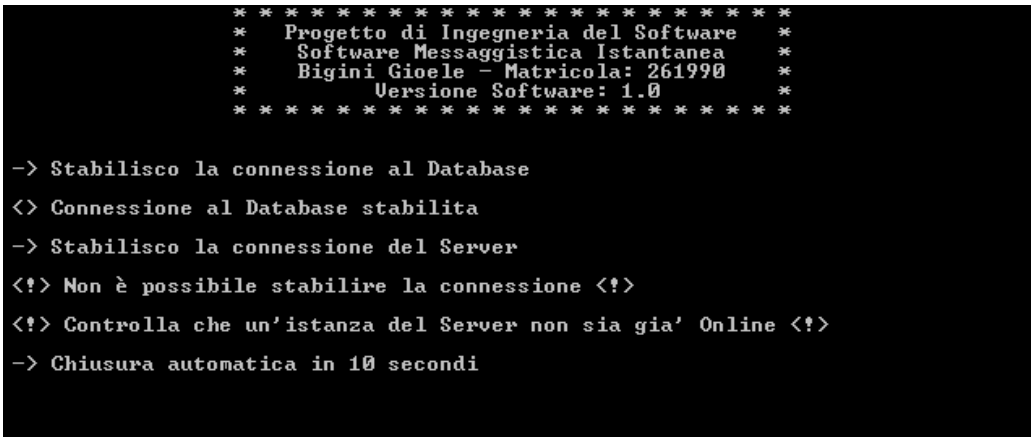
```
        var msgPack = new Pacchetto("ListaAmici", string.Format("{0}", username));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }

    /* Richiesta di accesso ai Servizi di messaggistica */
    public void login(Connessione collegamento,string username,string password)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
        var msgPack = new Pacchetto("Login", string.Format("{0},{1}",username,password));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }

    /* Richiesta di Registrazione al servizio */
    public void registrazione(Connessione collegamento, string username, string password)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
        var msgPack = new Pacchetto("Registrazione", string.Format("{0},{1}", username, password));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }

    /* Richiesta di richieste in Segreteria */
    public void segreteria(Connessione collegamento, string username)
    {
      /* Se siamo attualmente connessi al Server */
      if (collegamento.connessioneTCP.Connected)
      {
        /* Creazione del pacchetto */
        var msgPack = new Pacchetto("Segreteria", string.Format("{0}", username));
        /* Invio del messaggio */
        collegamento.InviaPacchetto(msgPack);
      }
    }
  }
}
```

Per la validazione del progetto si è scelto di utilizzare la tecnica BlackBox, mettendo in campo la tecnica WhiteBox solo nel momento in cui la tecnica BlackBox possa risultare non sufficientemente efficace.

BLACKBOX SERVER

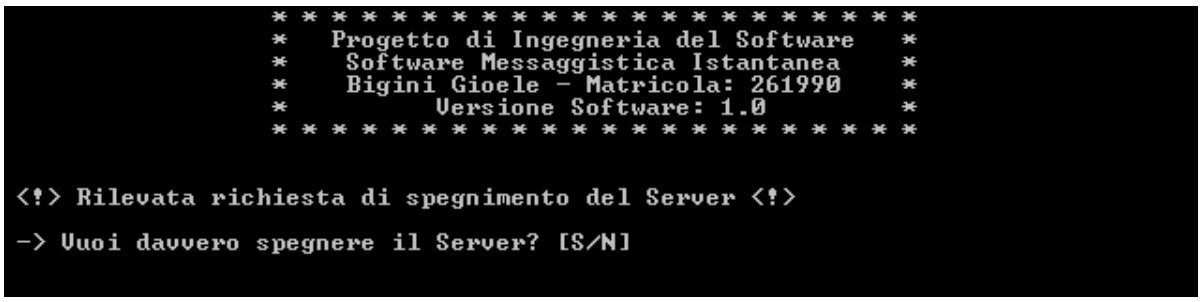
Schermata Iniziale

Errore avvio di istanza multipla del Server

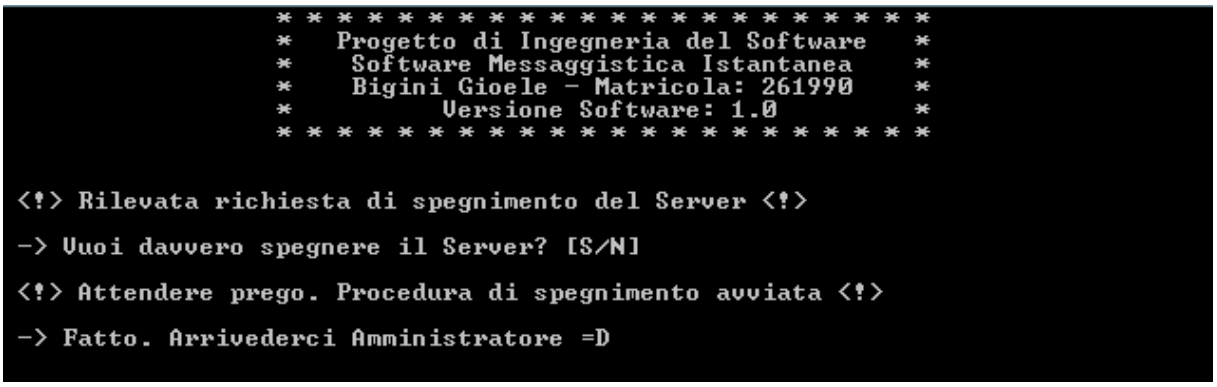


Monitor Eventi – Connessioni e Disconnessioni

Spegnimento

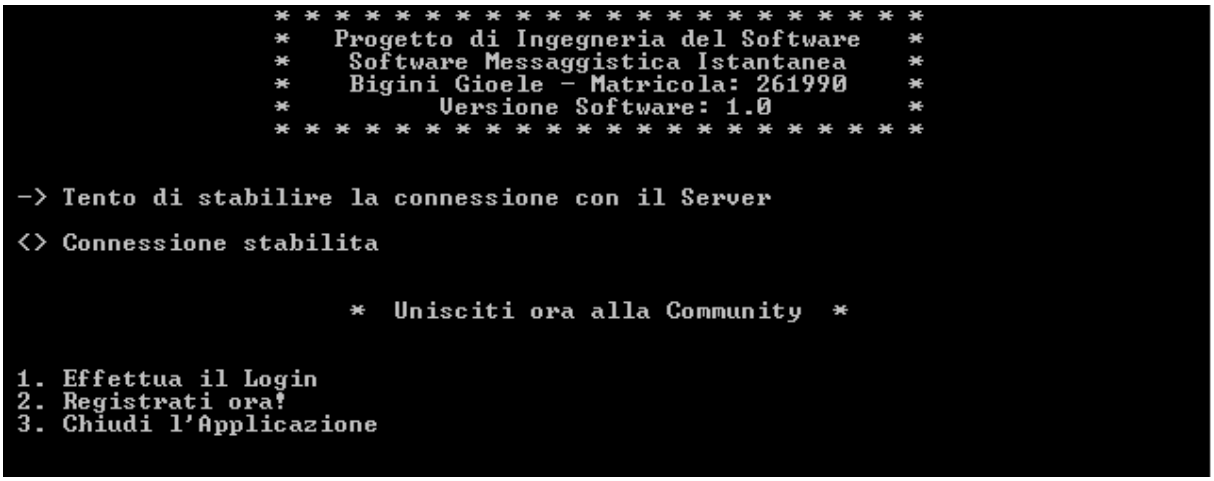


Conferma Spegnimento

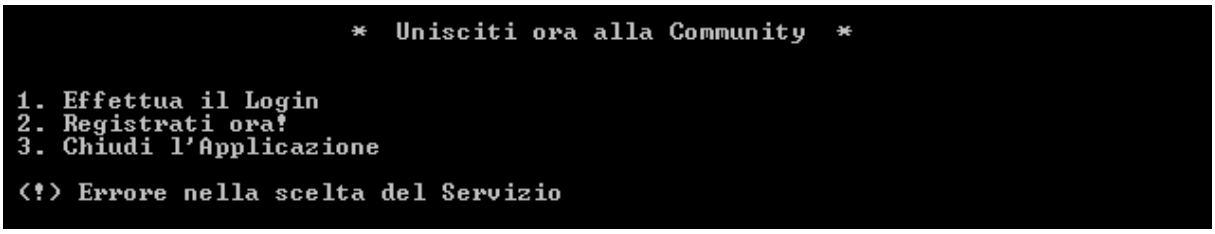


BLACKBOX CLIENT

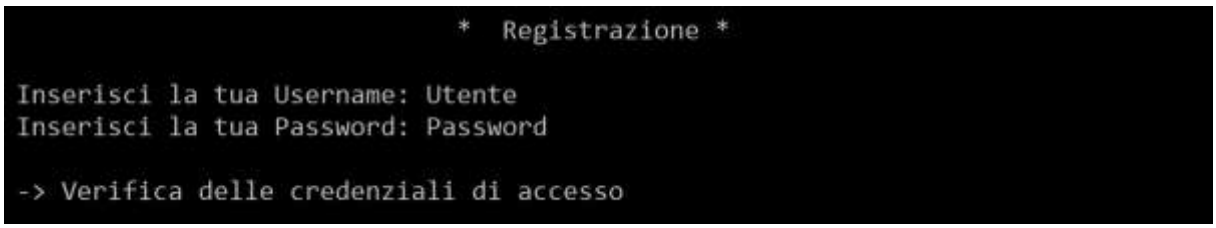
Schermata di Apertura



Errore nella scelta del Servizio



Registrazione Utente



Registrazione Fallita per Username Utilizzato

```
                          *  Registrazione *

Inserisci la tua Username: Utente
Inserisci la tua Password: Password

-> Verifica delle credenziali di accesso

[ERRORE] La Username immessa non e' disponibile
```

Registrazione Avvenuta con Successo

```
                          *  Registrazione *

Inserisci la tua Username: Utente
Inserisci la tua Password: Password

-> Verifica delle credenziali di accesso

[OK] Sei registrato. Ora puoi effettuare il Login
```

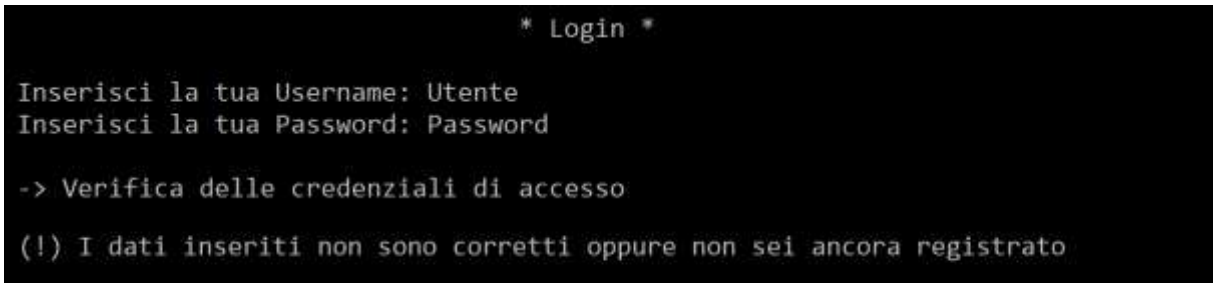
Login Utente

```
                              * Login *

Inserisci la tua Username: Utente
Inserisci la tua Password: Password

-> Verifica delle credenziali di accesso
```

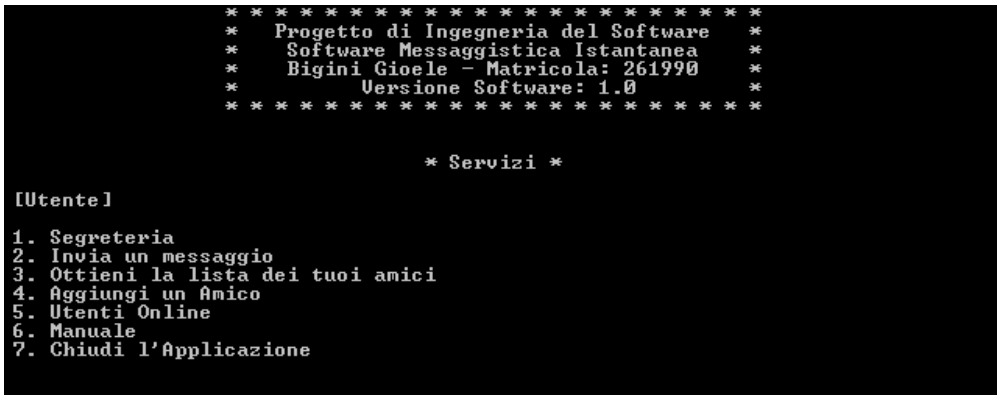
Login Fallito per connessione in altra posizione

```
                              * Login *

Inserisci la tua Username: Utente
Inserisci la tua Password: Password

-> Verifica delle credenziali di accesso

[ERRORE] Il tuo username e' gia' loggato in un altra posizione
```

Login Fallito per dati non validi



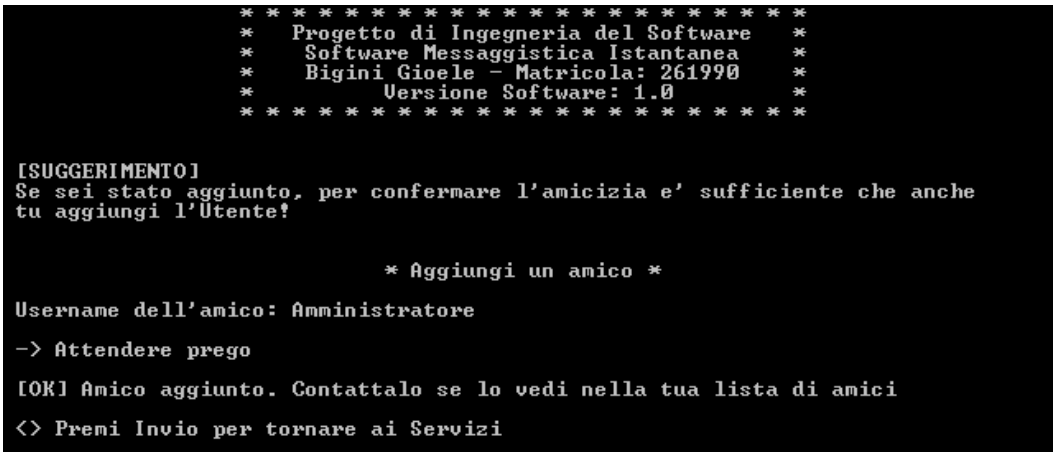
Login avvenuto con successo - ***Schermata dei Servizi***



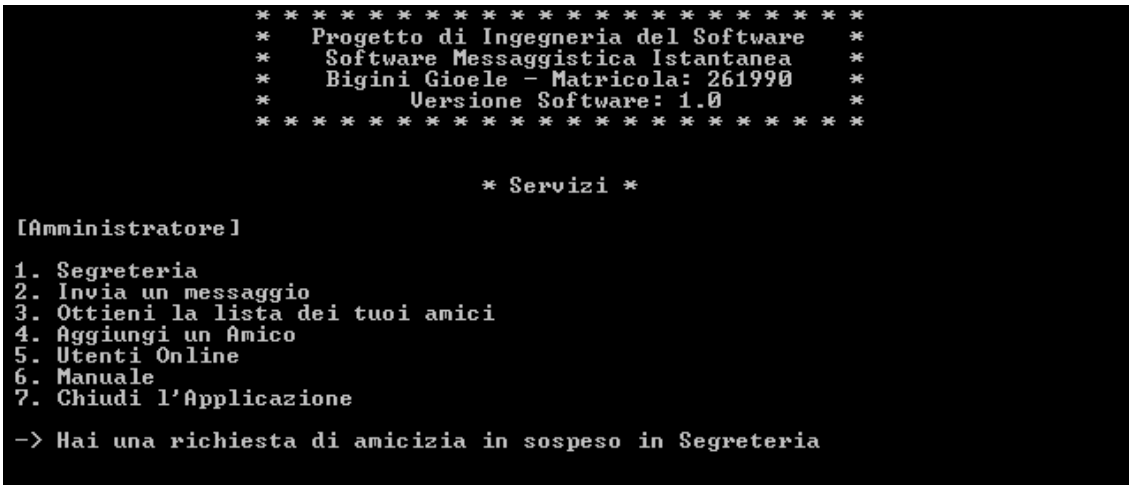
Servizio – Ottieni la lista dei tuoi amici – Lista Vuota

Servizio – Ottieni la lista dei tuoi amici – Lista non Vuota

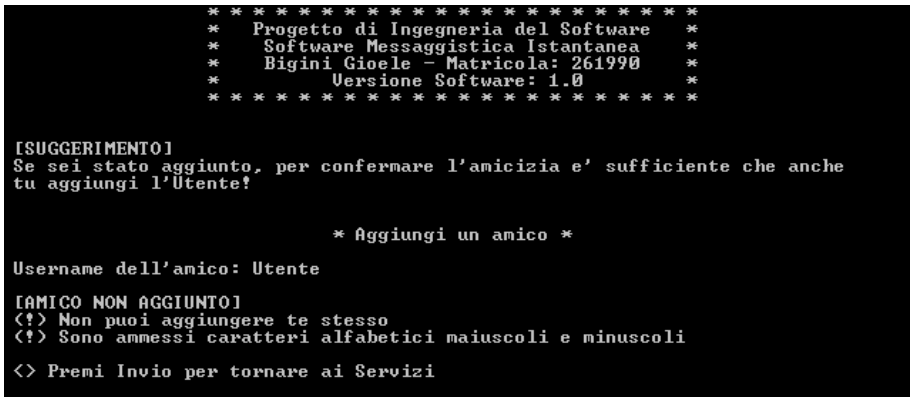
Servizio – Aggiungi un amico – Amico Aggiunto



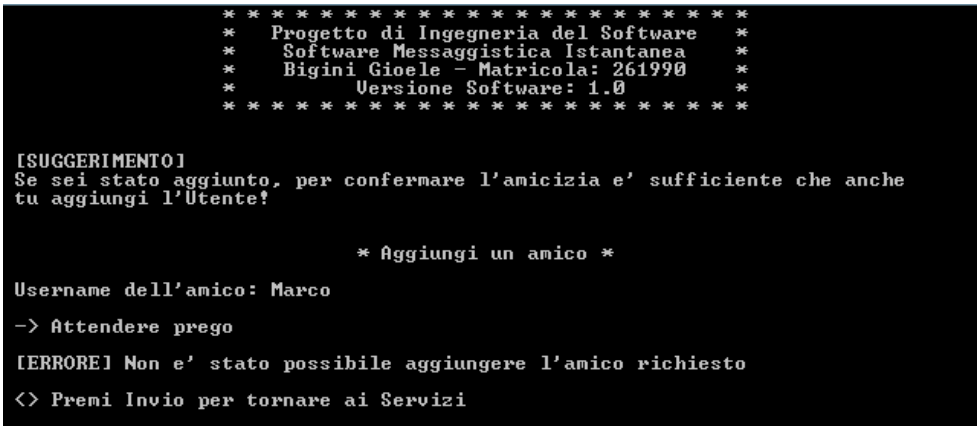
Servizio – Aggiungi un amico – Ricezione dell'amicizia



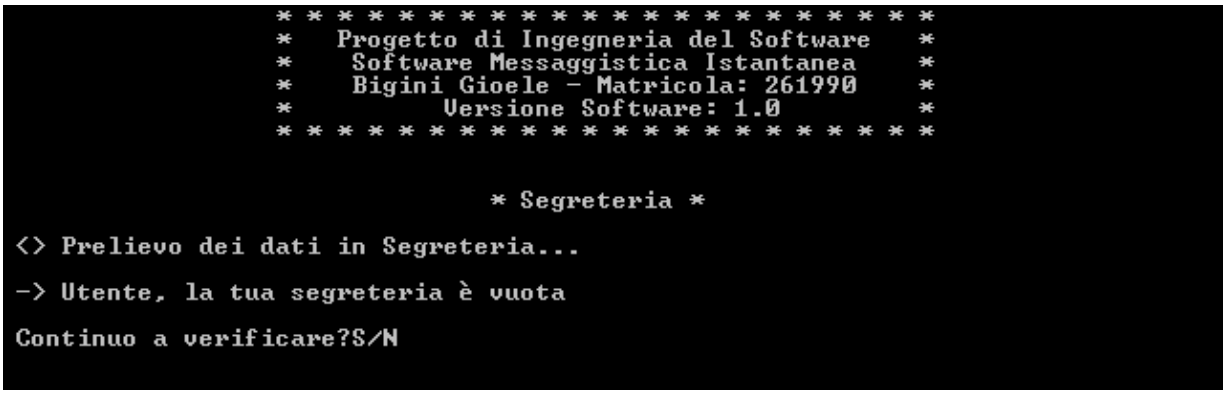
Servizio – Aggiungi un amico – Errore input invalido



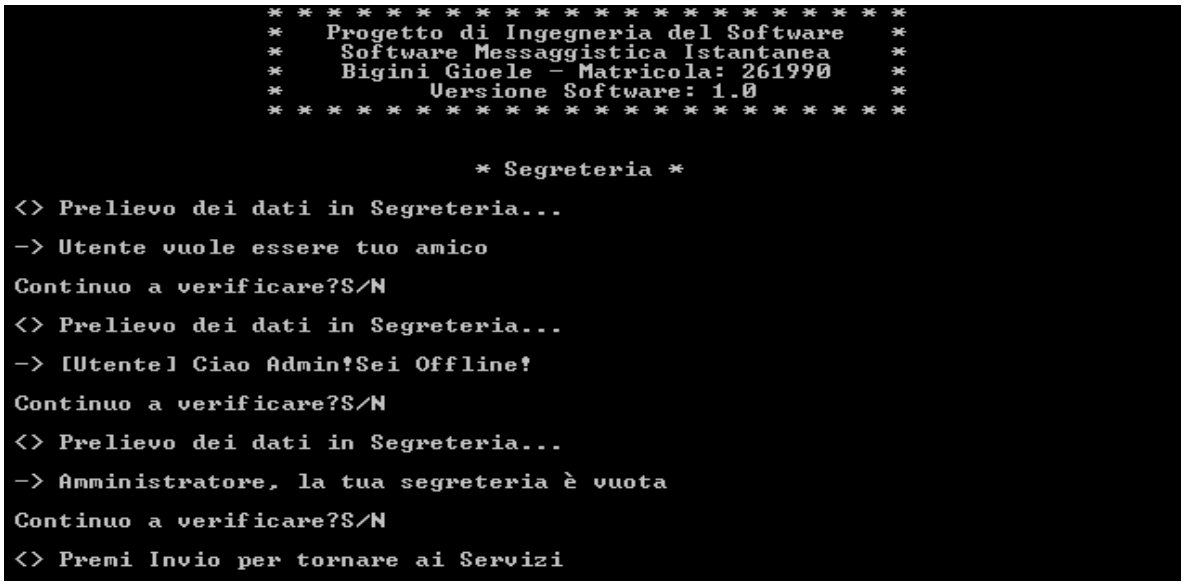
Servizio – Aggiungi un amico – Errore utente non esistente



Servizio – Segreteria – Segreteria Vuota

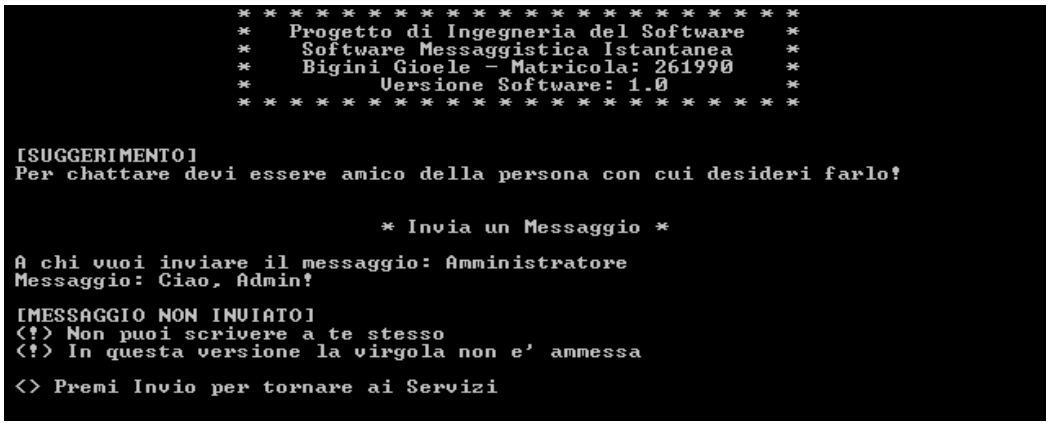


Servizio – Segreteria – Segreteria non Vuota

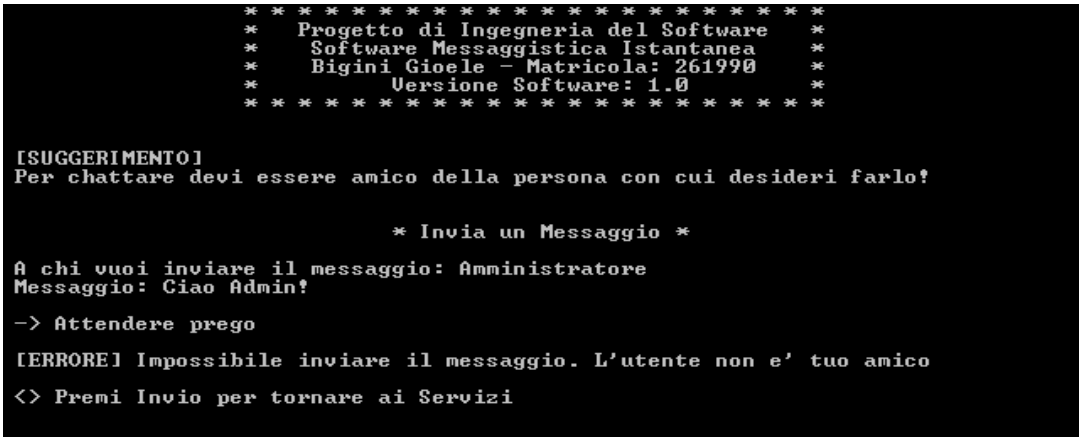


Servizio – Invia Messaggio - Messaggio Inviato

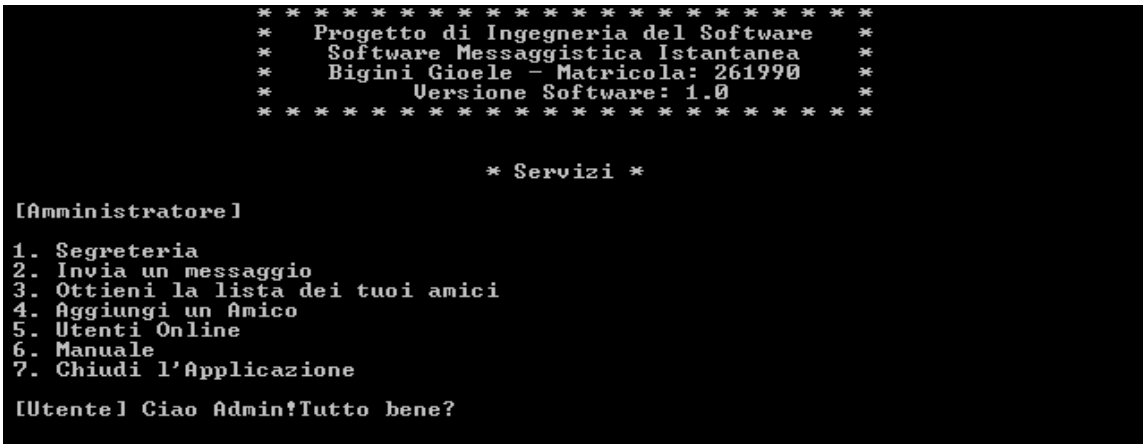
Servizio – Invia Messaggio – Errore caratteri invalidi



Servizio – Invia Messaggio
Errore di Invio messaggio a Utente non amico



Servizio – Invia Messaggio – Ricezione del Messaggio



Segnalazioni Utenti

Una serie di utenti dopo aver apprezzato l'applicazione ha criticato in maniera decisa l'usabilità. Le critiche hanno riguardato la travagliata comprensione del funzionamento dei servizi in assenza di un manuale e la impossibile conoscenza degli utenti online con cui poter comunicare. Essendo queste due funzionalità attese, esse hanno provocato una forte insoddisfazione per cui necessitano di essere aggiunte prima del rilascio finale. Alcuni utenti hanno addirittura aggiunto che in una fase di primo rilascio preferirebbero avere un software non proprio stabile piuttosto che un software che manca di funzionalità attese.

Manuale

Codice aggiunto in Interfaccia.cs di Client

```
          /* 6. Aiuto */
          case ConsoleKey.D6:
              /* Interruzione del Server */
              controllaCoda.Abort();
              /* Acquisizione dati da utente */
              Console.Clear();
              testoApertura();
              Console.WriteLine("\t\t\t\t  * Manuale *\n");

              Console.WriteLine(" [Chattare]");
              Console.WriteLine(" Per chattare devi essere amico della persona con cui desideri
farlo.");
              Console.WriteLine(" Per avere amici devi aggiungerli mediante - Aggiungi un Amico - .");
              Console.WriteLine(" Inviare messaggi e' semplice! Seleziona - Invia Messaggio - e
potrai");
              Console.WriteLine(" subito scrivere ai tuoi amici. I messaggi vengono inviati anche a
chi");
              Console.WriteLine(" e' offline. Gli verranno recapitati una volta online!\n");

              Console.WriteLine(" [Ricezione Messaggi]");
              Console.WriteLine(" I messaggi in arrivo vengono mostrati nella schermata Servizi. Se");
              Console.WriteLine(" usi un servizio non verra' mostrato alcun messaggio o
richiesta\n\n");

              Console.WriteLine(" [Aggiungere un amico]");
              Console.WriteLine(" Per aggiungere un amico è sufficiente utilizzare - Aggiungi un
Amico.");
              Console.WriteLine(" Non conosci nessuno? Utenti Online - ti mostrera' gli Utenti in
rete.");
              Console.WriteLine(" Sappi che non e' necessario che l'utente sia Online per aggiungerlo.
");
              Console.WriteLine(" Se invece vieni aggiunto, per confermare l'amicizia e' sufficiente");
              Console.WriteLine(" che anche tu aggiungi l'Utente, cosi' l'amicizia diventerà reale.");
              Console.WriteLine(" Ricorda! Con - Lista Utenti - sai in qualsiasi momento chi sono");
              Console.WriteLine(" i tuoi amici.\n");

              /* Rimanda ai Servizi se l'utente preme invio */
              Console.WriteLine("\n <> Premi Invio per tornare ai Servizi\n");
              Console.ReadLine();
              Console.Clear();
              testoServizi(username);

              /* Reset della pressione */
```

```
            pressione = new ConsoleKeyInfo();

            /* Riavvio Thread chiamate asincrone */
            controllaCoda = new Thread(stampaChiamataAsincrona);
            controllaCoda.Start();

            break;
```

Test Client

Lista Utenti Online

Nuovo Servizio ListaUtenti.cs in Server

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server.Servizi
{
  class ListaUtenti : Servizio
  {
    /* Servizio per recuperare la lista di amici */
    override public void forniciServizio(Ricettore client, List<string> lista)
    {
      /* Istanza di convertitore */
      Convertitore converti = new Convertitore();
      /* Variabile predisposta per contenere la lista degli amici */
      List<string> listaUtentiOnline = new List<string>();
```

```
      /* Recupero Client Connessi */
      /* Variabile per la copia di Dictionary per effettuare elaborazioni locali al metodo */
      Dictionary<Ricettore, string> clientConnessi;
      lock(Connessione.Istanza.ClientConnessi)
        clientConnessi = Connessione.Istanza.ClientConnessi;

      /* Costruzione della lista di utenti Online (viene escluso il richiedente) */
      foreach (KeyValuePair<Ricettore, string> utente in clientConnessi)
        listaUtentiOnline.Add(utente.Value);

      /* Invia la lista di amici */
      Pacchetto messaggio = new Pacchetto("ListaUtentiOnline",
                                          converti.listaAStringa(listaUtentiOnline));
      client.InviaPacchetto(messaggio);
    }
  }
}
```

Codice aggiunto in Ricettore.cs di Server

```
    /* Metodo per l'elaborazione della richiesta ricevuta dal Client Remoto */
    private void elaboraRichiesta(string data)
    {
      try
      {
        /* Lettura del comando */
        var pacchetto = new Pacchetto(data);
        string comando = pacchetto.Comando;

        /* Conversione del messaggio in lista */
        Convertitore converti = new Convertitore();
        var lista = converti.stringaALista(pacchetto.Contenuto);

        /* Smistamento della richiesta al servizio */
        switch (comando)
        {
          case "Login":
            new Login().forniciServizio(this, lista);
            break;
          case "Registrazione":
            new Registrazione().forniciServizio(this, lista);
            break;
          case "AggiungiAmico":
            new Amicizia().forniciServizio(this, lista);
            break;
          case "Messaggio":
            new Messaggistica().forniciServizio(this, lista);
            break;
          case "ListaAmici":
            new Lista().forniciServizio(this, lista);
            break;
          case "ListaUtentiOnline":
            new ListaUtenti().forniciServizio(this, lista);
            break;
          case "Segreteria":
            new Segreteria().forniciServizio(this, lista);
            break;
          default:
            Interfaccia.stampaMessaggio("Pacchetto Invalido");
            break;
        }
      }
      catch (Exception)
      {
        Interfaccia.stampaMessaggio(" (!) Il pacchetto ricevuto è Invalido\n");
      }
    }
```

Codice aggiunto in Interfaccia.cs di Client

```
        /* 5. Utenti Online */
        case ConsoleKey.D5:
            /* Interruzione del Server */
            controllaCoda.Abort();
            /* Acquisizione dati da utente */
            Console.Clear();
            testoApertura();
            Console.WriteLine("\t\t\t       * Utenti Online *\n");

            /* Invia la richiesta */
            richiesta.listaUtentiOnline(collegamento, username);

            /* Attendi la risposta del Server */
            Console.WriteLine(" <> Recupero della lista di utenti Online...\n");
            mutex.WaitOne();

            try
            {
              /* Scrivi la risposta a schermo */
              lista = converti.StringaALista(Connessione.Istanza.chiamataSincrona);

              foreach (string utente in lista)
                if(utente.Length > 0)
                  Console.WriteLine(" -> {0}", utente);
            }
            catch (Exception)
            {
              Console.WriteLine(" (!) Si e' verificato un errore nella richiesta. Riprova\n");
              lista = new List<string>();
            }

            /* Rimanda ai Servizi se l'utente preme invio */
            Console.WriteLine("\n <> Premi Invio per tornare ai Servizi\n");
            Console.ReadLine();
            Console.Clear();
            testoServizi(username);

            /* Reset della pressione */
            pressione = new ConsoleKeyInfo();

            /* Riavvio Thread chiamate asincrone */
            controllaCoda = new Thread(stampaChiamataAsincrona);
            controllaCoda.Start();
            break;
```

Codice aggiunto Connessione.cs di Client

```
/* Metodo per l'elaborazione dei messaggi ricevuti dal Server */
private void elaboraRichiesta(string richiesta)
{
  /* Generazione di un pacchetto interpretabile */
  Pacchetto pacchetto = new Pacchetto(richiesta);
  /* Estrazione del comando */
  string comando = pacchetto.Comando;

  /* Comportamento relativo al comando ricevuto */
  switch (comando)
  {
    /* Chiamate sincrone */
    case "Login":
    case "Registrazione":
    case "AggiungiAmico":
    case "Messaggio":
    case "ListaAmici":
    case "ListaUtentiOnline":
    case "Segreteria":
        chiamataSincrona = pacchetto.Contenuto;
        Interfaccia.liberaMutex();
```

```
            break;

        /* Chiamate asincrone */
        case "InoltroAmicizia":
        case "InoltroMsg":
            chiamateAsincrone.Enqueue(pacchetto);
            break;

        /* Se il server spedisce pacchetti anomali */
        default:
            Interfaccia.stampaErrore("Il server ti ha inviato un pacchetto anomalo");
            break;
    }
}
```

Codice aggiunto Richiesta.cs di Client

```
/* Richiesta ottenimento lista amici */
public void listaUtentiOnline(Connessione collegamento, string username)
{
  /* Se siamo attualmente connessi al Server */
  if (collegamento.connessioneTCP.Connected)
  {
    /* Creazione del pacchetto */
    var msgPack = new Pacchetto("ListaUtentiOnline", string.Format("{0}", username));
    /* Invio del messaggio */
    collegamento.InviaPacchetto(msgPack);
  }
}
```

Test Client

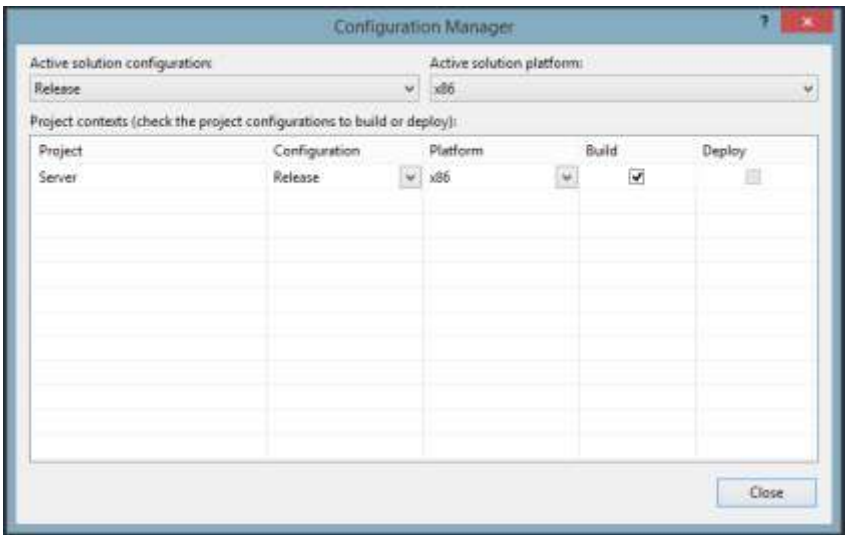
```
* * * * * * * * * * * * * * * * * * * * * * *
*   Progetto di Ingegneria del Software     *
*    Software Messaggistica Istantanea      *
*    Bigini Gioele - Matricola: 261990      *
*          Versione Software: 1.0           *
* * * * * * * * * * * * * * * * * * * * * * *


                    * Utenti Online *

<> Recupero della lista di utenti Online...

-> Utente

<> Premi Invio per tornare ai Servizi
```

L'ambiente di sviluppo utilizzato è **Microsoft Visual Studio 2013 Community Edition**.
Il Framework di sviluppo è **Microsoft .NET Framework 4.6.**

- Il Server è stato sviluppato per CPU a 32 bit (x86) coerentemente alla **DLL** di SQLite, sviluppata per sistemi a 32 bit.
- Il Client è stato sviluppato per supportare qualsiasi CPU.

Sarà possibile compilare le due applicazioni come segue:

1. Estrarre la cartella progetto_OOP_bigini.zip
2. Aprire la soluzione del progetto SoluzioneServer.sln con Microsoft Visual Studio 2013 Community
3. In "Esplora Soluzioni" di Visual Studio, cliccare con il tasto destro sul progetto Client, quindi cliccare in primo luogo su Pulisci (o Clean) e in seguito Compila (o Build). Assicurarsi che l'applicazione venga compilata per CPU x86. Se così non fosse, in caso di Warning emesso da parte di Visual Studio, recarsi nella barra degli strumenti superiore, quindi cliccare sul menù a tendina Compila (Build) e cercare la voce "Configuration Manager", quindi impostare come nello screen seguente:



   Il messaggio di Warning scomparirà e si potrà ricompilare il progetto.
4. Aprire la soluzione del progetto SoluzioneClient.sln con Microsoft Visual Studio 2013 Community
5. In "Esplora Soluzioni" di Visual Studio, cliccare con il tasto destro sul progetto Server, quindi cliccare in primo luogo su Pulisci (o Clean) e in seguito Compila (o Build).
6. Navigare con Esplora Risorse in:

progetto_OOPSE_bigini ***/soluzioni_progetto/Server/bin/x86/Release***

7. Doppio click sull'eseguibile:

   Server.exe

8. Navigare con Esplora Risorse in:

   progetto_OOPSE_bigini ***/soluzioni_progetto/Client/bin/Release***

9. Doppio click sull'eseguibile:

   Client.exe

È possibile avviare contemporaneamente più applicazioni Client.
È possibile avviare una sola applicazione Server.
Perchè il sistema funzioni, è necessario avviare prima il Server.

All'avvio del Server viene creato nel Database SQLite un Utente di default con le credenziali seguenti:

Username	***Password***
Amministratore	Password

Per resettare il Database è sufficiente cancellare il file "database.sqlite" in:

progetto_OOPSE_bigini***/soluzioni_progetto/Server/bin/x86/Release***

Il Server si connette al proprio indirizzo speciale in rete, 127.0.0.1 comunemente detto localhost. La porta, 5302, è arbitraria, pertanto se si ha necessità di cambiarla perchè già occupata è necessario modificare delle linee di codice in Server e in Client.
Precisamente nella soluzione di Server è possibile effettuare questa modifica nella classe Connessione.cs, alla riga 31 e 32.
Precisamente nella soluzione di Client è possibile effettuare questa modifica nella classe Connessione.cs alla riga 33.

Il software è stato collaudato sulle seguenti macchine con hardware e sistema operativo differenti:

SO	RAM	Core	Architettura
Windows 10	4	2	x64
Windows 8.1	8	4	x64